Freiman's $3k - 4$ Theorem

Arthur F. Ramos David Barros Hulak Ruy J. G. B. de Queiroz

July 7, 2026

Abstract

This entry formalizes Freiman's $3k - 4$ theorem for finite sets of integers: small doubling, in the range $|A + A| \leq 3|A| - 4$, forces containment in a short arithmetic progression. AI assistance was used for proof engineering. The final definitions, statements, and proofs are checked by Isabelle.

1 Overview

Freiman's $3k - 4$ theorem is a classical inverse theorem in additive combinatorics. It gives a sharp structural conclusion for finite integer sets whose two-fold sumset is only slightly larger than the Cauchy-Davenport lower bound: if A has cardinality $k \geq 3$ and $|A + A| \leq 3k - 4$, then A is contained in an arithmetic progression of length at most $|A + A| - k + 1$.

2 Formalization

The development starts with integer arithmetic progressions, affine normalization of finite integer sets, and cardinality invariance of sumsets under injective affine maps. The final proof combines these reductions with the additive-combinatorial core argument.

3 Sources

The formalization follows the standard presentation of Freiman's theorem in additive-combinatorics texts, especially Nathanson [1] and Tao and Vu [2].

Contents

```
theory Freiman-Sumset-Basics
  imports Main
begin

definition sumset :: ('a::comm-monoid-add) set ⇒ 'a set ⇒ 'a set where
  sumset A B = {x. ∃ a∈A. ∃ b∈B. x = a + b}

lemma sumset-iff:
  x ∈ sumset A B ⟷ (∃ a∈A. ∃ b∈B. x = a + b)
  by (auto simp: sumset-def)

lemma sumsetI [intro]:
  assumes a ∈ A b ∈ B
  shows a + b ∈ sumset A B
  using assms by (auto simp: sumset-def)

lemma sumsetE [elim]:
  assumes x ∈ sumset A B
  obtains a b where a ∈ A b ∈ B x = a + b
  using assms by (auto simp: sumset-def)

lemma sumset-as-image:
  sumset A B = case-prod (+) ` (A × B)
  by (auto simp: sumset-def)

lemma sumset-commute:
  sumset A B = sumset B A
proof
  show sumset A B ⊆ sumset B A
  proof
    fix x
    assume x ∈ sumset A B
    then obtain a b where a ∈ A b ∈ B x = a + b
      by (auto simp: sumset-def)
    then show x ∈ sumset B A
    proof −
      have b + a ∈ sumset B A
        using ‹b ∈ B› ‹a ∈ A› by auto
      then show ?thesis
        using ‹x = a + b› by (simp add: add.commute)
    qed
  qed
next
  show sumset B A ⊆ sumset A B
  proof
    fix x
    assume x ∈ sumset B A
    then obtain b a where b ∈ B a ∈ A x = b + a
      by (auto simp: sumset-def)
    then show x ∈ sumset A B
    proof −
      have a + b ∈ sumset A B
        using ‹a ∈ A› ‹b ∈ B› by auto
      then show ?thesis
        using ‹x = b + a› by (simp add: add.commute)
    qed
  qed
qed
```

```
lemma empty-sumset-left [simp]:
  sumset {} B = {}
  by (auto simp: sumset-def)

lemma empty-sumset-right [simp]:
  sumset A {} = {}
  by (auto simp: sumset-def)

lemma sumset-assoc:
  fixes A B C :: ('a::comm-monoid-add) set
  shows sumset (sumset A B) C = sumset A (sumset B C)
proof
  show sumset (sumset A B) C ⊆ sumset A (sumset B C)
  proof
    fix x
    assume x ∈ sumset (sumset A B) C
    then obtain ab c where ab: ab ∈ sumset A B and c: c ∈ C and x: x = ab + c
      by blast
    from ab obtain a b where a: a ∈ A and b: b ∈ B and ab-eq: ab = a + b
      by blast
    have b + c ∈ sumset B C
      using b c by auto
    then show x ∈ sumset A (sumset B C)
    proof −
      have a + (b + c) ∈ sumset A (sumset B C)
        using a ‹b + c ∈ sumset B C› by auto
      then show ?thesis
        using x ab-eq by (simp add: add.assoc)
    qed
  qed
next
  show sumset A (sumset B C) ⊆ sumset (sumset A B) C
  proof
    fix x
    assume x ∈ sumset A (sumset B C)
    then obtain a bc where a: a ∈ A and bc: bc ∈ sumset B C and x: x = a + bc
      by blast
    from bc obtain b c where b: b ∈ B and c: c ∈ C and bc-eq: bc = b + c
      by blast
    have a + b ∈ sumset A B
      using a b by auto
    then show x ∈ sumset (sumset A B) C
    proof −
      have (a + b) + c ∈ sumset (sumset A B) C
      proof (rule sumsetI)
        show a + b ∈ sumset A B
          using ‹a + b ∈ sumset A B› .
        show c ∈ C
          using c .
      qed
      then show ?thesis
        using x bc-eq by (simp add: add.assoc)
    qed
  qed
qed

lemma finite-sumset [intro]:
  assumes finite A finite B
  shows finite (sumset A B)
```

unfolding *sumset-as-image*
using *assms* **by** (*intro finite-imageI finite-cartesian-product*)

end
theory *Freiman-3k-4*
imports
Freiman-Sumset-Basics
HOL−Computational-Algebra.Group-Closure
Kneser-Cauchy-Davenport.Kneser-Cauchy-Davenport-main-proofs
begin

4 Freiman's $3k - 4$ theorem for integer sumsets

This theory develops the integer-side infrastructure for Freiman's $3k - 4$ theorem. The final theorem is most naturally stated after normalizing a finite integer set by translation and dilation; the lemmas below record the affine invariance and interval containment facts used by that reduction.

4.1 Integer arithmetic progressions

definition *int-ap-segment* :: *int* ⇒ *int* ⇒ *nat* ⇒ *int set* **where**
int-ap-segment a d n = (λ*i*. *a* + *int i* ∗ *d*) ‘ {..<*n*}

definition *int-arithmetic-progression* :: *int set* ⇒ *bool* **where**
int-arithmetic-progression A ⟷ (∃ *a d n*. *A* = *int-ap-segment a d n*)

lemma *int-arithmetic-progressionI*:
A = *int-ap-segment a d n* ⟹ *int-arithmetic-progression A*
unfolding *int-arithmetic-progression-def* **by** *blast*

lemma *int-arithmetic-progressionE*:
assumes *int-arithmetic-progression A*
obtains *a d n* **where** *A* = *int-ap-segment a d n*
using *assms* **unfolding** *int-arithmetic-progression-def* **by** *blast*

lemma *finite-int-ap-segment* [*simp*]:
finite (*int-ap-segment a d n*)
by (*simp add*: *int-ap-segment-def*)

lemma *int-ap-segment-empty* [*simp*]:
int-ap-segment a d 0 = {}
by (*simp add*: *int-ap-segment-def*)

lemma *inj-on-int-ap-segment-index*:
assumes *d* ≠ *0*
shows *inj-on* (λ*i*. *a* + *int i* ∗ *d*) {..<*n*}
proof (*rule inj-onI*)
fix *i j*
assume *i*: *i* ∈ {..<*n*} **and** *j*: *j* ∈ {..<*n*}
assume *eq*: *a* + *int i* ∗ *d* = *a* + *int j* ∗ *d*
from *eq* **have** *int i* ∗ *d* = *int j* ∗ *d*
by *simp*
with *assms* **have** *int i* = *int j*
by *simp*
then show *i* = *j*
by *simp*
qed

lemma *card-int-ap-segment*:

assumes $d \neq 0$
shows *card* (*int-ap-segment a d n*) = *n*
using *assms*
by (*simp add*: *int-ap-segment-def card-image inj-on-int-ap-segment-index*)

lemma *int-ap-segment-one-eq-atLeastAtMost*:
assumes $a \leq b$
shows *int-ap-segment a 1* (*nat* (*b* − *a* + *1*)) = {*a*..*b*}
proof
show *int-ap-segment a 1* (*nat* (*b* − *a* + *1*)) ⊆ {*a*..*b*}
proof
fix *x*
assume *x* ∈ *int-ap-segment a 1* (*nat* (*b* − *a* + *1*))
then obtain *i* **where** *i-lt*: *i* < *nat* (*b* − *a* + *1*) **and** *x*: *x* = *a* + *int i*
by (*auto simp*: *int-ap-segment-def*)
from *assms* **have** *0* < *b* − *a* + *1*
by *linarith*
with *i-lt* **have** *int i* < *b* − *a* + *1*
by *linarith*
with *x* **show** *x* ∈ {*a*..*b*}
by *auto*
qed
next
show {*a*..*b*} ⊆ *int-ap-segment a 1* (*nat* (*b* − *a* + *1*))
proof
fix *x*
assume *x-in*: *x* ∈ {*a*..*b*}
then have *ax-nonneg*: *0* ≤ *x* − *a*
by *simp*
define *i* **where** *i* = *nat* (*x* − *a*)
from *ax-nonneg* **have** *int-i*: *int i* = *x* − *a*
by (*simp add*: *i-def*)
from *x-in* **have** *x* − *a* < *b* − *a* + *1*
by *simp*
with *int-i* **have** *i* < *nat* (*b* − *a* + *1*)
by *linarith*
moreover have *x* = *a* + *int i*
using *int-i* **by** *simp*
ultimately show *x* ∈ *int-ap-segment a 1* (*nat* (*b* − *a* + *1*))
by (*auto simp*: *int-ap-segment-def*)
qed
qed

lemma *finite-int-set-subset-min-max-ap*:
assumes *fin*: *finite A* **and** *nonempty*: *A* ≠ {}
shows *A* ⊆ *int-ap-segment* (*Min A*) *1* (*nat* (*Max A* − *Min A* + *1*))
proof −
have *Min A* ≤ *Max A*
using *assms* **by** *simp*
then have *interval-eq*: *int-ap-segment* (*Min A*) *1* (*nat* (*Max A* − *Min A* + *1*)) = {*Min A*..*Max A*}
by (*rule int-ap-segment-one-eq-atLeastAtMost*)
have *A* ⊆ {*Min A*..*Max A*}
using *assms* **by** *auto*
then show *?thesis*
using *interval-eq* **by** *simp*
qed

4.2 Affine images and sumsets

definition *affine-image-int* :: *int* ⇒ *int* ⇒ *int set* ⇒ *int set* **where**
affine-image-int c d A = (λ*x*. *c* + *d* ∗ *x*) ‘ *A*

lemma *affine-image-int-iff*:
x ∈ *affine-image-int c d A* ⟷ (∃ *a*∈*A*. *x* = *c* + *d* ∗ *a*)
by (*auto simp*: *affine-image-int-def*)

lemma *finite-affine-image-int* [*intro*]:
assumes *finite A*
shows *finite* (*affine-image-int c d A*)
using *assms* **by** (*simp add*: *affine-image-int-def*)

lemma *inj-on-affine-image-int*:
fixes *c d* :: *int*
assumes *d* ≠ *0*
shows *inj-on* (λ*x*. *c* + *d* ∗ *x*) *A*
proof (*rule inj-onI*)
fix *x y*
assume *x* ∈ *A* **and** *y* ∈ *A*
assume *eq*: *c* + *d* ∗ *x* = *c* + *d* ∗ *y*
have (*c* + *d* ∗ *x*) − *c* = (*c* + *d* ∗ *y*) − *c*
using *eq* **by** *simp*
then have *d* ∗ *x* = *d* ∗ *y*
by *simp*
with *assms* **show** *x* = *y*
by *simp*
qed

lemma *card-affine-image-int*:
assumes *fin*: *finite A* **and** *d-nonzero*: *d* ≠ *0*
shows *card* (*affine-image-int c d A*) = *card A*
using *fin inj-on-affine-image-int*[*OF d-nonzero*, *of c A*]
by (*simp add*: *affine-image-int-def card-image*)

lemma *affine-image-int-sumset*:
sumset (*affine-image-int c d A*) (*affine-image-int e d B*) =
affine-image-int (*c* + *e*) *d* (*sumset A B*)
proof
show *sumset* (*affine-image-int c d A*) (*affine-image-int e d B*) ⊆
affine-image-int (*c* + *e*) *d* (*sumset A B*)
proof
fix *x*
assume *x* ∈ *sumset* (*affine-image-int c d A*) (*affine-image-int e d B*)
then obtain *y z* **where**
y: *y* ∈ *affine-image-int c d A*
and *z*: *z* ∈ *affine-image-int e d B*
and *x*: *x* = *y* + *z*
by (*rule sumsetE*)
from *y* **obtain** *a* **where** *a*: *a* ∈ *A* **and** *y-eq*: *y* = *c* + *d* ∗ *a*
by (*auto simp*: *affine-image-int-def*)
from *z* **obtain** *b* **where** *b*: *b* ∈ *B* **and** *z-eq*: *z* = *e* + *d* ∗ *b*
by (*auto simp*: *affine-image-int-def*)
have *a* + *b* ∈ *sumset A B*
using *a b* **by** (*rule sumsetI*)
moreover have *x* = (*c* + *e*) + *d* ∗ (*a* + *b*)
using *x y-eq z-eq* **by** (*simp add*: *algebra-simps*)
ultimately show *x* ∈ *affine-image-int* (*c* + *e*) *d* (*sumset A B*)

```
    by (auto simp: affine-image-int-def)
  qed
next
  show affine-image-int (c + e) d (sumset A B) ⊆
      sumset (affine-image-int c d A) (affine-image-int e d B)
  proof
    fix x
    assume x ∈ affine-image-int (c + e) d (sumset A B)
    then obtain s where s: s ∈ sumset A B and x: x = (c + e) + d * s
      by (auto simp: affine-image-int-def)
    from s obtain a b where a: a ∈ A and b: b ∈ B and s-eq: s = a + b
      by (rule sumsetE)
    have c + d * a ∈ affine-image-int c d A
      using a by (auto simp: affine-image-int-def)
    moreover have e + d * b ∈ affine-image-int e d B
      using b by (auto simp: affine-image-int-def)
    moreover have x = (c + d * a) + (e + d * b)
      using x s-eq by (simp add: algebra-simps)
    ultimately show x ∈ sumset (affine-image-int c d A) (affine-image-int e d B)
      by (metis sumsetI)
  qed
qed

lemma affine-image-int-sumset-self:
  sumset (affine-image-int c d A) (affine-image-int c d A) =
    affine-image-int (2 * c) d (sumset A A)
proof −
  have sumset (affine-image-int c d A) (affine-image-int c d A) =
      affine-image-int (c + c) d (sumset A A)
    by (rule affine-image-int-sumset)
  also have ... = affine-image-int (2 * c) d (sumset A A)
    by simp
  finally show ?thesis .
qed

lemma card-sumset-affine-image-int:
  assumes finA: finite A and finB: finite B and d-nonzero: d ≠ 0
  shows card (sumset (affine-image-int c d A) (affine-image-int e d B)) =
    card (sumset A B)
proof −
  have card (sumset (affine-image-int c d A) (affine-image-int e d B)) =
      card (affine-image-int (c + e) d (sumset A B))
    by (simp add: affine-image-int-sumset)
  also have ... = card (sumset A B)
    by (rule card-affine-image-int[OF finite-sumset[OF finA finB] d-nonzero])
  finally show ?thesis .
qed

lemma card-sumset-affine-image-int-self:
  assumes fin: finite A and d-nonzero: d ≠ 0
  shows card (sumset (affine-image-int c d A) (affine-image-int c d A)) =
    card (sumset A A)
  by (rule card-sumset-affine-image-int[OF fin fin d-nonzero])

lemma affine-image-int-zero-one [simp]:
  affine-image-int 0 1 A = A
  by (auto simp: affine-image-int-def)
```

4.3 Endpoint lower bound for two-fold sumsets

lemma *endpoint-affine-images-inter*:
fixes $A :: int\ set$
assumes *fin*: *finite* A **and** *nonempty*: $A \neq \{\}$
shows *affine-image-int* $(Min\ A)\ 1\ A \cap$ *affine-image-int* $(Max\ A)\ 1\ A =$
$\{Min\ A + Max\ A\}$
proof
show *affine-image-int* $(Min\ A)\ 1\ A \cap$ *affine-image-int* $(Max\ A)\ 1\ A \subseteq$
$\{Min\ A + Max\ A\}$
proof
fix x
assume *x-in*: $x \in$ *affine-image-int* $(Min\ A)\ 1\ A \cap$ *affine-image-int* $(Max\ A)\ 1\ A$
then obtain a **where** *a-in*: $a \in A$ **and** *x-left*: $x = Min\ A + a$
by (*auto simp*: *affine-image-int-def*)
from *x-in* **obtain** b **where** *b-in*: $b \in A$ **and** *x-right*: $x = Max\ A + b$
by (*auto simp*: *affine-image-int-def*)
have *a-le*: $a \leq Max\ A$
using *fin a-in* **by** *simp*
have *min-le-b*: $Min\ A \leq b$
using *fin b-in* **by** *simp*
have $x \leq Min\ A + Max\ A$
using *x-left a-le* **by** *simp*
moreover have $Min\ A + Max\ A \leq x$
using *x-right min-le-b* **by** *simp*
ultimately have $x = Min\ A + Max\ A$
by *simp*
then show $x \in \{Min\ A + Max\ A\}$
by *simp*
qed
next
have *min-in*: $Min\ A \in A$
using *fin nonempty* **by** *simp*
have *max-in*: $Max\ A \in A$
using *fin nonempty* **by** *simp*
show $\{Min\ A + Max\ A\} \subseteq$
affine-image-int $(Min\ A)\ 1\ A \cap$ *affine-image-int* $(Max\ A)\ 1\ A$
using *min-in max-in* **by** (*auto simp*: *affine-image-int-def add.commute*)
qed

lemma *endpoint-affine-union-card*:
fixes $A :: int\ set$
assumes *fin*: *finite* A **and** *nonempty*: $A \neq \{\}$
shows *card* (*affine-image-int* $(Min\ A)\ 1\ A \cup$ *affine-image-int* $(Max\ A)\ 1\ A) =$
$2 * card\ A - 1$
proof −
let $?L =$ *affine-image-int* $(Min\ A)\ 1\ A$
let $?R =$ *affine-image-int* $(Max\ A)\ 1\ A$
have *finL*: *finite* $?L$
by (*rule finite-affine-image-int*[*OF fin*])
have *finR*: *finite* $?R$
by (*rule finite-affine-image-int*[*OF fin*])
have *cardL*: *card* $?L =$ *card* A
by (*rule card-affine-image-int*[*OF fin*]) *simp*
have *cardR*: *card* $?R =$ *card* A
by (*rule card-affine-image-int*[*OF fin*]) *simp*
have *inter*: $?L \cap ?R = \{Min\ A + Max\ A\}$
by (*rule endpoint-affine-images-inter*[*OF fin nonempty*])
have *card* $?L +$ *card* $?R =$ *card* $(?L \cup ?R) +$ *card* $(?L \cap ?R)$

```
    by (rule card-Un-Int[OF finL finR])
  then have card A + card A = card (?L ∪ ?R) + 1
    using cardL cardR inter by simp
  moreover have 0 < card A
    using fin nonempty by (simp add: card-gt-0-iff)
  ultimately show ?thesis
    by simp
qed

lemma endpoint-affine-images-inter-two-sets:
  fixes A B :: int set
  assumes finA: finite A and nonemptyA: A ≠ {}
  assumes finB: finite B and nonemptyB: B ≠ {}
  shows affine-image-int (Min A) 1 B ∩ affine-image-int (Max B) 1 A =
    {Min A + Max B}
proof
  show affine-image-int (Min A) 1 B ∩ affine-image-int (Max B) 1 A ⊆
      {Min A + Max B}
  proof
    fix x
    assume x-in: x ∈ affine-image-int (Min A) 1 B ∩ affine-image-int (Max B) 1 A
    then obtain b where b-in: b ∈ B and x-left: x = Min A + b
      by (auto simp: affine-image-int-def)
    from x-in obtain a where a-in: a ∈ A and x-right: x = Max B + a
      by (auto simp: affine-image-int-def)
    have b-le: b ≤ Max B
      using finB b-in by simp
    have min-le-a: Min A ≤ a
      using finA a-in by simp
    have x ≤ Min A + Max B
      using x-left b-le by simp
    moreover have Min A + Max B ≤ x
      using x-right min-le-a by simp
    ultimately have x = Min A + Max B
      by simp
    then show x ∈ {Min A + Max B}
      by simp
  qed
next
  have min-in: Min A ∈ A
    using finA nonemptyA by simp
  have max-in: Max B ∈ B
    using finB nonemptyB by simp
  show {Min A + Max B} ⊆
      affine-image-int (Min A) 1 B ∩ affine-image-int (Max B) 1 A
    using min-in max-in by (auto simp: affine-image-int-def add.commute)
qed

lemma card-sumset-ge-card-add-card-minus-one:
  fixes A B :: int set
  assumes finA: finite A and finB: finite B
  assumes nonemptyA: A ≠ {} and nonemptyB: B ≠ {}
  shows card A + card B − 1 ≤ card (sumset A B)
proof −
  let ?L = affine-image-int (Min A) 1 B
  let ?R = affine-image-int (Max B) 1 A
  have min-in: Min A ∈ A
    using finA nonemptyA by simp
  have max-in: Max B ∈ B
```

```
    using finB nonemptyB by simp
  have finL: finite ?L
    by (rule finite-affine-image-int[OF finB])
  have finR: finite ?R
    by (rule finite-affine-image-int[OF finA])
  have cardL: card ?L = card B
    by (rule card-affine-image-int[OF finB]) simp
  have cardR: card ?R = card A
    by (rule card-affine-image-int[OF finA]) simp
  have inter: ?L ∩ ?R = {Min A + Max B}
    by (rule endpoint-affine-images-inter-two-sets[OF finA nonemptyA finB nonemptyB])
  have card ?L + card ?R = card (?L ∪ ?R) + card (?L ∩ ?R)
    by (rule card-Un-Int[OF finL finR])
  then have card-union: card (?L ∪ ?R) = card A + card B − 1
    using cardL cardR inter finA nonemptyA by simp
  have union-subset: ?L ∪ ?R ⊆ sumset A B
  proof
    fix x
    assume x-in: x ∈ ?L ∪ ?R
    then show x ∈ sumset A B
    proof
      assume x ∈ ?L
      then obtain b where b-in: b ∈ B and x-eq: x = Min A + b
        by (auto simp: affine-image-int-def)
      then show ?thesis
        using min-in by (auto intro: sumsetI)
    next
      assume x ∈ ?R
      then obtain a where a-in: a ∈ A and x-eq: x = Max B + a
        by (auto simp: affine-image-int-def)
      have a + Max B ∈ sumset A B
        using a-in max-in by (rule sumsetI)
      then show ?thesis
        using x-eq by (simp add: add.commute)
    qed
  qed
  have card (?L ∪ ?R) ≤ card (sumset A B)
    by (rule card-mono[OF finite-sumset[OF finA finB] union-subset])
  with card-union show ?thesis
    by simp
qed

lemma card-sumset-self-ge-two-card-minus-one:
  fixes A :: int set
  assumes fin: finite A and nonempty: A ≠ {}
  shows 2 * card A − 1 ≤ card (sumset A A)
proof −
  let ?L = affine-image-int (Min A) 1 A
  let ?R = affine-image-int (Max A) 1 A
  have min-in: Min A ∈ A
    using fin nonempty by simp
  have max-in: Max A ∈ A
    using fin nonempty by simp
  have finL: finite ?L
    by (rule finite-affine-image-int[OF fin])
  have finR: finite ?R
    by (rule finite-affine-image-int[OF fin])
  have card-union: card (?L ∪ ?R) = 2 * card A − 1
    by (rule endpoint-affine-union-card[OF fin nonempty])
```

```
  have union-subset: ?L ∪ ?R ⊆ sumset A A
  proof
    fix x
    assume x-in: x ∈ ?L ∪ ?R
    then show x ∈ sumset A A
    proof
      assume x ∈ ?L
      then obtain a where a-in: a ∈ A and x-eq: x = Min A + a
        by (auto simp: affine-image-int-def)
      then show ?thesis
        using min-in by (auto intro: sumsetI)
    next
      assume x ∈ ?R
      then obtain a where a-in: a ∈ A and x-eq: x = Max A + a
        by (auto simp: affine-image-int-def)
      then show ?thesis
        using max-in by (auto intro: sumsetI)
    qed
  qed
  have card (?L ∪ ?R) ≤ card (sumset A A)
    by (rule card-mono[OF finite-sumset[OF fin fin] union-subset])
  with card-union show ?thesis
    by simp
qed
```

4.4 Holes in the normalized interval

```
definition interval-holes :: int set ⇒ int set where
  interval-holes A = {x. 0 ≤ x ∧ x ≤ Max A ∧ x ∉ A}

definition lower-sum-holes :: int set ⇒ int set where
  lower-sum-holes A = {x ∈ interval-holes A. x ∈ sumset A A}

definition upper-sum-holes :: int set ⇒ int set where
  upper-sum-holes A = {x ∈ interval-holes A. Max A + x ∈ sumset A A}

definition stable-sum-holes :: int set ⇒ int set where
  stable-sum-holes A =
    interval-holes A − (lower-sum-holes A ∪ upper-sum-holes A)

definition left-stable-sum-holes :: int set ⇒ int set where
  left-stable-sum-holes A = interval-holes A − lower-sum-holes A

definition right-stable-sum-holes :: int set ⇒ int set where
  right-stable-sum-holes A = interval-holes A − upper-sum-holes A

lemma finite-interval-holes [simp]:
  finite (interval-holes A)
proof −
  have interval-holes A ⊆ {0..Max A}
    by (auto simp: interval-holes-def)
  then show ?thesis
    by (rule finite-subset) simp
qed

lemma finite-lower-sum-holes [simp]:
  finite (lower-sum-holes A)
  unfolding lower-sum-holes-def by simp
```

lemma *finite-upper-sum-holes* [*simp*]:
finite (*upper-sum-holes A*)
unfolding *upper-sum-holes-def* **by** *simp*

lemma *finite-stable-sum-holes* [*simp*]:
finite (*stable-sum-holes A*)
unfolding *stable-sum-holes-def* **by** *simp*

lemma *finite-left-stable-sum-holes* [*simp*]:
finite (*left-stable-sum-holes A*)
unfolding *left-stable-sum-holes-def* **by** *simp*

lemma *finite-right-stable-sum-holes* [*simp*]:
finite (*right-stable-sum-holes A*)
unfolding *right-stable-sum-holes-def* **by** *simp*

lemma *stable-sum-holes-eq-left-right-inter*:
stable-sum-holes A =
left-stable-sum-holes A ∩ *right-stable-sum-holes A*
by (*auto simp*: *stable-sum-holes-def left-stable-sum-holes-def right-stable-sum-holes-def*)

lemma *left-stable-sum-hole-notin-sumset*:
assumes *x-left*: *x* ∈ *left-stable-sum-holes A*
shows *x* ∈ *interval-holes A x* ∉ *sumset A A*
proof −
show *x-hole*: *x* ∈ *interval-holes A*
using *x-left* **by** (*simp add*: *left-stable-sum-holes-def*)
show *x* ∉ *sumset A A*
proof
assume *x* ∈ *sumset A A*
with *x-hole* **have** *x* ∈ *lower-sum-holes A*
by (*simp add*: *lower-sum-holes-def*)
with *x-left* **show** *False*
by (*simp add*: *left-stable-sum-holes-def*)
qed
qed

lemma *right-stable-sum-hole-notin-sumset*:
assumes *x-right*: *x* ∈ *right-stable-sum-holes A*
shows *x* ∈ *interval-holes A Max A* + *x* ∉ *sumset A A*
proof −
show *x-hole*: *x* ∈ *interval-holes A*
using *x-right* **by** (*simp add*: *right-stable-sum-holes-def*)
show *Max A* + *x* ∉ *sumset A A*
proof
assume *Max A* + *x* ∈ *sumset A A*
with *x-hole* **have** *x* ∈ *upper-sum-holes A*
by (*simp add*: *upper-sum-holes-def*)
with *x-right* **show** *False*
by (*simp add*: *right-stable-sum-holes-def*)
qed
qed

lemma *left-stable-hole-prefix-card-le-holes*:
fixes *A* :: *int set*
assumes *fin*: *finite A*
assumes *x-left*: *x* ∈ *left-stable-sum-holes A*
shows *card* (*A* ∩ {*0*..*x*}) ≤ *card* ({*0*..*x*} − *A*)
proof −

```
  let ?P = A ∩ {0..x}
  let ?Q = {0..x} − A
  let ?f = λa. x − a
  have x-not-sum: x ∉ sumset A A
    by (rule left-stable-sum-hole-notin-sumset(2)[OF x-left])
  have image-sub: ?f ‘ ?P ⊆ ?Q
  proof
    fix y
    assume y ∈ ?f ‘ ?P
    then obtain a where a-in: a ∈ A and a-bounds: 0 ≤ a a ≤ x and y-eq: y = x − a
      by auto
    have y-bounds: 0 ≤ y y ≤ x
      using a-bounds y-eq by auto
    have y ∉ A
    proof
      assume y-in: y ∈ A
      have a + y ∈ sumset A A
        using a-in y-in by (rule sumsetI)
      moreover have a + y = x
        using y-eq by simp
      ultimately show False
        using x-not-sum by simp
    qed
    with y-bounds show y ∈ ?Q
      by simp
  qed
  have inj: inj-on ?f ?P
    by (rule inj-onI) simp
  have finP: finite ?P
    using fin by simp
  have card ?P = card (?f ‘ ?P)
    by (simp add: card-image finP inj)
  also have ... ≤ card ?Q
    by (rule card-mono[OF - image-sub]) simp
  finally show ?thesis .
qed

lemma left-stable-hole-prefix-twice-card-le:
  fixes A :: int set
  assumes fin: finite A
  assumes x-left: x ∈ left-stable-sum-holes A
  shows 2 * card (A ∩ {0..x}) ≤ nat (x + 1)
proof −
  let ?P = A ∩ {0..x}
  let ?Q = {0..x} − A
  have x-hole: x ∈ interval-holes A
    by (rule left-stable-sum-hole-notin-sumset(1)[OF x-left])
  then have x-nonneg: 0 ≤ x
    by (simp add: interval-holes-def)
  have P-sub: ?P ⊆ {0..x}
    by simp
  have Q-eq: ?Q = {0..x} − ?P
    by auto
  have cardQ: card ?Q = card {0..x} − card ?P
    by (simp add: Q-eq card-Diff-subset[OF - P-sub])
  have prefix-le: card ?P ≤ card ?Q
    by (rule left-stable-hole-prefix-card-le-holes[OF fin x-left])
  have P-card-le: card ?P ≤ card {0..x}
    by (rule card-mono[OF - P-sub]) simp
```

```
  have 2 * card ?P ≤ card {0..x}
    using prefix-le cardQ P-card-le by linarith
  also have ... = nat (x + 1)
    using x-nonneg by simp
  finally show ?thesis .
qed

lemma right-stable-hole-suffix-card-le-holes:
  fixes A :: int set
  assumes fin: finite A
  assumes x-right: x ∈ right-stable-sum-holes A
  shows card (A ∩ {x..Max A}) ≤ card ({x..Max A} - A)
proof -
  let ?n = Max A
  let ?P = A ∩ {x..?n}
  let ?Q = {x..?n} - A
  let ?f = λa. ?n + x - a
  have nx-not-sum: ?n + x ∉ sumset A A
    by (rule right-stable-sum-hole-notin-sumset(2)[OF x-right])
  have image-sub: ?f ` ?P ⊆ ?Q
  proof
    fix y
    assume y ∈ ?f ` ?P
    then obtain a where a-in: a ∈ A and a-bounds: x ≤ a a ≤ ?n
      and y-eq: y = ?n + x - a
      by auto
    have y-bounds: x ≤ y y ≤ ?n
      using a-bounds y-eq by auto
    have y ∉ A
    proof
      assume y-in: y ∈ A
      have a + y ∈ sumset A A
        using a-in y-in by (rule sumsetI)
      moreover have a + y = ?n + x
        using y-eq by simp
      ultimately show False
        using nx-not-sum by simp
    qed
    with y-bounds show y ∈ ?Q
      by simp
  qed
  have inj: inj-on ?f ?P
    by (rule inj-onI) simp
  have finP: finite ?P
    using fin by simp
  have card ?P = card (?f ` ?P)
    by (simp add: card-image finP inj)
  also have ... ≤ card ?Q
    by (rule card-mono[OF - image-sub]) simp
  finally show ?thesis .
qed

lemma right-stable-hole-suffix-twice-card-le:
  fixes A :: int set
  assumes fin: finite A
  assumes x-right: x ∈ right-stable-sum-holes A
  shows 2 * card (A ∩ {x..Max A}) ≤ nat (Max A - x + 1)
proof -
  let ?n = Max A
```

```
  let ?P = A ∩ {x..?n}
  let ?Q = {x..?n} − A
  have x-hole: x ∈ interval-holes A
    by (rule right-stable-sum-hole-notin-sumset(1)[OF x-right])
  then have x-le: x ≤ ?n
    by (simp add: interval-holes-def)
  have P-sub: ?P ⊆ {x..?n}
    by simp
  have Q-eq: ?Q = {x..?n} − ?P
    by auto
  have cardQ: card ?Q = card {x..?n} − card ?P
    by (simp add: Q-eq card-Diff-subset[OF - P-sub])
  have suffix-le: card ?P ≤ card ?Q
    by (rule right-stable-hole-suffix-card-le-holes[OF fin x-right])
  have P-card-le: card ?P ≤ card {x..?n}
    by (rule card-mono[OF - P-sub]) simp
  have 2 * card ?P ≤ card {x..?n}
    using suffix-le cardQ P-card-le by linarith
  also have ... = nat (?n − x + 1)
    using x-le by simp
  finally show ?thesis .
qed

lemma hole-cover-of-no-stable-sum-holes:
  assumes stable-empty: stable-sum-holes A = {}
  shows card (interval-holes A) ≤ card (lower-sum-holes A) + card (upper-sum-holes A)
proof −
  have subset: interval-holes A ⊆ lower-sum-holes A ∪ upper-sum-holes A
    using stable-empty by (auto simp: stable-sum-holes-def)
  have card (interval-holes A) ≤ card (lower-sum-holes A ∪ upper-sum-holes A)
    by (rule card-mono[OF finite-UnI[OF finite-lower-sum-holes finite-upper-sum-holes] subset])
  also have ... ≤ card (lower-sum-holes A) + card (upper-sum-holes A)
    by (rule card-Un-le)
  finally show ?thesis .
qed

lemma interval-holes-eq-interval-diff:
  assumes subset: A ⊆ {0..Max A}
  shows interval-holes A = {0..Max A} − A
  using subset by (auto simp: interval-holes-def)

lemma normalized-subset-interval:
  fixes A :: int set
  assumes fin: finite A
  assumes zero: 0 ∈ A
  assumes nonneg: ⋀x. x ∈ A ⟹ 0 ≤ x
  shows A ⊆ {0..Max A}
proof
  fix x
  assume x-in: x ∈ A
  have 0 ≤ x
    by (rule nonneg[OF x-in])
  moreover have x ≤ Max A
    using fin x-in by simp
  ultimately show x ∈ {0..Max A}
    by simp
qed

lemma card-interval-holes:
```

fixes $A :: int\ set$
assumes *fin*: *finite A*
assumes *zero*: $0 \in A$
assumes *nonneg*: $\bigwedge x.\ x \in A \Longrightarrow 0 \leq x$
shows *card* (*interval-holes A*) = *nat* (*Max A* + *1*) − *card A*
proof −
have *subset*: $A \subseteq \{0..Max\ A\}$
by (*rule normalized-subset-interval*[*OF fin zero nonneg*])
have *max-nonneg*: $0 \leq Max\ A$
using *fin zero* **by** *simp*
have *card* (*interval-holes A*) = *card* ($\{0..Max\ A\}$ − *A*)
by (*simp add*: *interval-holes-eq-interval-diff*[*OF subset*])
also have ... = *card* $\{0..Max\ A\}$ − *card A*
by (*rule card-Diff-subset*[*OF fin subset*])
also have ... = *nat* (*Max A* + *1*) − *card A*
using *max-nonneg* **by** *simp*
finally show *?thesis* **.**
qed

lemma *Min-eq-zero-of-zero-mem-nonneg*:
assumes *fin*: *finite A*
assumes *zero*: $0 \in A$
assumes *nonneg*: $\bigwedge x.\ x \in A \Longrightarrow 0 \leq x$
shows *Min A* = *0*
proof (*rule antisym*)
show *Min A* ≤ *0*
using *fin zero* **by** *simp*
next
have $A \neq \{\}$
using *zero* **by** *auto*
then have *Min A* ∈ *A*
using *fin* **by** *simp*
then show *0* ≤ *Min A*
by (*rule nonneg*)
qed

lemma *normalized-endpoint-union-card*:
fixes $A :: int\ set$
assumes *fin*: *finite A*
assumes *zero*: $0 \in A$
assumes *nonneg*: $\bigwedge x.\ x \in A \Longrightarrow 0 \leq x$
shows *card* (*A* ∪ *affine-image-int* (*Max A*) *1 A*) = *2* ∗ *card A* − *1*
proof −
have *nonempty*: $A \neq \{\}$
using *zero* **by** *auto*
have *min0*: *Min A* = *0*
by (*rule Min-eq-zero-of-zero-mem-nonneg*[*OF fin zero nonneg*])
have *card* (*affine-image-int* (*Min A*) *1 A* ∪ *affine-image-int* (*Max A*) *1 A*) =
2 ∗ *card A* − *1*
by (*rule endpoint-affine-union-card*[*OF fin nonempty*])
then show *?thesis*
by (*simp add*: *min0*)
qed

lemma *lower-sum-holes-disjoint-endpoint-union*:
fixes $A :: int\ set$
assumes *fin*: *finite A*
assumes *zero*: $0 \in A$
assumes *nonneg*: $\bigwedge x.\ x \in A \Longrightarrow 0 \leq x$

```
  shows (A ∪ affine-image-int (Max A) 1 A) ∩ lower-sum-holes A = {}
proof
  show (A ∪ affine-image-int (Max A) 1 A) ∩ lower-sum-holes A ⊆ {}
  proof
    fix x
    assume x-in: x ∈ (A ∪ affine-image-int (Max A) 1 A) ∩ lower-sum-holes A
    then have x-hole: x ∈ interval-holes A
      by (simp add: lower-sum-holes-def)
    then have x-notin-A: x ∉ A and x-le: x ≤ Max A
      by (auto simp: interval-holes-def)
    from x-in have x-base: x ∈ A ∪ affine-image-int (Max A) 1 A
      by simp
    then show x ∈ {}
    proof
      assume x ∈ A
      with x-notin-A show ?thesis
        by simp
    next
      assume x ∈ affine-image-int (Max A) 1 A
      then obtain a where a-in: a ∈ A and x-eq: x = Max A + a
        by (auto simp: affine-image-int-def)
      have 0 ≤ a
        by (rule nonneg[OF a-in])
      with x-eq x-le have a0: a = 0
        by simp
      have Max A ∈ A
        using fin zero Max-in by fastforce
      with x-notin-A x-eq a0 show ?thesis
        by simp
    qed
  qed
qed simp

lemma upper-sum-holes-image-disjoint-endpoint-union:
  fixes A :: int set
  assumes fin: finite A
  assumes zero: 0 ∈ A
  assumes nonneg: ⋀x. x ∈ A ⟹ 0 ≤ x
  shows (A ∪ affine-image-int (Max A) 1 A) ∩
    affine-image-int (Max A) 1 (upper-sum-holes A) = {}
proof
  show (A ∪ affine-image-int (Max A) 1 A) ∩
      affine-image-int (Max A) 1 (upper-sum-holes A) ⊆ {}
  proof
    fix y
    assume y-in: y ∈ (A ∪ affine-image-int (Max A) 1 A) ∩
        affine-image-int (Max A) 1 (upper-sum-holes A)
    then obtain x where x-upper: x ∈ upper-sum-holes A and y-eq: y = Max A + x
      by (auto simp: affine-image-int-def)
    then have x-hole: x ∈ interval-holes A
      by (simp add: upper-sum-holes-def)
    then have x-nonneg: 0 ≤ x and x-notin-A: x ∉ A
      by (auto simp: interval-holes-def)
    from y-in have y-base: y ∈ A ∪ affine-image-int (Max A) 1 A
      by simp
    then show y ∈ {}
    proof
      assume yA: y ∈ A
      have y-le: y ≤ Max A
```

```
      using fin yA by simp
    with y-eq x-nonneg have x0: x = 0
      by simp
    with zero x-notin-A show ?thesis
      by simp
  next
    assume y ∈ affine-image-int (Max A) 1 A
    then obtain a where a-in: a ∈ A and ya: y = Max A + a
      by (auto simp: affine-image-int-def)
    from y-eq ya have x = a
      by simp
    with x-notin-A a-in show ?thesis
      by simp
  qed
 qed
qed simp

lemma lower-sum-holes-disjoint-upper-sum-holes-image:
 fixes A :: int set
 assumes fin: finite A
 assumes zero: 0 ∈ A
 assumes nonneg: ⋀x. x ∈ A ⟹ 0 ≤ x
 shows lower-sum-holes A ∩ affine-image-int (Max A) 1 (upper-sum-holes A) = {}
proof
 show lower-sum-holes A ∩ affine-image-int (Max A) 1 (upper-sum-holes A) ⊆ {}
 proof
  fix y
  assume y-in: y ∈ lower-sum-holes A ∩ affine-image-int (Max A) 1 (upper-sum-holes A)
  then have y-hole: y ∈ interval-holes A
    by (simp add: lower-sum-holes-def)
  then have y-le: y ≤ Max A and y-notin-A: y ∉ A
    by (auto simp: interval-holes-def)
  from y-in obtain x where x-upper: x ∈ upper-sum-holes A and y-eq: y = Max A + x
    by (auto simp: affine-image-int-def)
  then have x-hole: x ∈ interval-holes A
    by (simp add: upper-sum-holes-def)
  then have x-nonneg: 0 ≤ x and x-notin-A: x ∉ A
    by (auto simp: interval-holes-def)
  with y-eq y-le have x0: x = 0
    by simp
  with zero x-notin-A show y ∈ {}
    by simp
 qed
qed simp

lemma normalized-sumset-lower-bound-with-holes:
 fixes A :: int set
 assumes fin: finite A
 assumes zero: 0 ∈ A
 assumes nonneg: ⋀x. x ∈ A ⟹ 0 ≤ x
 shows 2 * card A - 1 + card (lower-sum-holes A) + card (upper-sum-holes A)
   ≤ card (sumset A A)
proof -
 let ?B = A ∪ affine-image-int (Max A) 1 A
 let ?L = lower-sum-holes A
 let ?U = affine-image-int (Max A) 1 (upper-sum-holes A)
 let ?S = ?B ∪ ?L ∪ ?U
 have nonempty: A ≠ {}
  using zero by auto
```

have *max-in*: *Max A* ∈ *A*
using *fin nonempty* **by** *simp*
have *finB*: *finite ?B*
by (*rule finite-UnI*[*OF fin finite-affine-image-int*[*OF fin*]])
have *finL*: *finite ?L*
by *simp*
have *finU*: *finite ?U*
by (*rule finite-affine-image-int*) *simp*
have *cardB*: *card ?B* = *2* ∗ *card A* − *1*
by (*rule normalized-endpoint-union-card*[*OF fin zero nonneg*])
have *cardU*: *card ?U* = *card* (*upper-sum-holes A*)
by (*rule card-affine-image-int*) *simp-all*
have *BL-disj*: *?B* ∩ *?L* = {}
by (*rule lower-sum-holes-disjoint-endpoint-union*[*OF fin zero nonneg*])
have *BU-disj*: *?B* ∩ *?U* = {}
by (*rule upper-sum-holes-image-disjoint-endpoint-union*[*OF fin zero nonneg*])
have *LU-disj*: *?L* ∩ *?U* = {}
by (*rule lower-sum-holes-disjoint-upper-sum-holes-image*[*OF fin zero nonneg*])
have *BL-card*: *card* (*?B* ∪ *?L*) = *card ?B* + *card ?L*
by (*rule card-Un-disjoint*[*OF finB finL BL-disj*])
have *BL-U-disj*: (*?B* ∪ *?L*) ∩ *?U* = {}
using *BU-disj LU-disj* **by** *auto*
have *cardS*: *card ?S* = *2* ∗ *card A* − *1* + *card ?L* + *card* (*upper-sum-holes A*)
proof −
have *card ?S* = *card* (*?B* ∪ *?L*) + *card ?U*
by (*rule card-Un-disjoint*[*OF finite-UnI*[*OF finB finL*] *finU BL-U-disj*])
also have . . . = *card ?B* + *card ?L* + *card ?U*
using *BL-card* **by** *simp*
also have . . . = *2* ∗ *card A* − *1* + *card ?L* + *card* (*upper-sum-holes A*)
using *cardB cardU* **by** *simp*
finally show *?thesis* **.**
qed
have *S-subset*: *?S* ⊆ *sumset A A*
proof
fix *y*
assume *y-in*: *y* ∈ *?S*
then show *y* ∈ *sumset A A*
proof
assume *yBL*: *y* ∈ *?B* ∪ *?L*
then show *?thesis*
proof
assume *yB*: *y* ∈ *?B*
then show *?thesis*
proof
assume *yA*: *y* ∈ *A*
have *0* + *y* ∈ *sumset A A*
proof (*rule sumsetI*)
show *0* ∈ *A*
by (*rule zero*)
show *y* ∈ *A*
by (*rule yA*)
qed
then show *?thesis*
by *simp*
next
assume *y* ∈ *affine-image-int* (*Max A*) *1 A*
then obtain *a* **where** *a-in*: *a* ∈ *A* **and** *y-eq*: *y* = *Max A* + *a*
by (*auto simp*: *affine-image-int-def*)
have *Max A* + *a* ∈ *sumset A A*

```
        using max-in a-in by (rule sumsetI)
      then show ?thesis
        using y-eq by simp
    qed
  next
    assume y ∈ ?L
    then show ?thesis
      by (simp add: lower-sum-holes-def)
  qed
next
  assume y ∈ ?U
  then obtain x where x-in: x ∈ upper-sum-holes A and y-eq: y = Max A + x
    by (auto simp: affine-image-int-def)
  then have Max A + x ∈ sumset A A
    by (simp add: upper-sum-holes-def)
  then show ?thesis
    using y-eq by simp
qed
qed
have card ?S ≤ card (sumset A A)
  by (rule card-mono[OF finite-sumset[OF fin fin] S-subset])
with cardS show ?thesis
  by simp
qed

lemma normalized-sumset-eq-endpoint-union-with-holes:
  fixes A :: int set
  assumes fin: finite A
  assumes zero: 0 ∈ A
  assumes nonneg: ⋀x. x ∈ A ⟹ 0 ≤ x
  shows sumset A A =
    A ∪ affine-image-int (Max A) 1 A ∪
    lower-sum-holes A ∪
    affine-image-int (Max A) 1 (upper-sum-holes A)
proof
  let ?n = Max A
  have nonempty: A ≠ {}
    using zero by auto
  have top: ?n ∈ A
    using fin nonempty by simp
  have subset: A ⊆ {0..?n}
    by (rule normalized-subset-interval[OF fin zero nonneg])
  show sumset A A ⊆
      A ∪ affine-image-int ?n 1 A ∪
      lower-sum-holes A ∪
      affine-image-int ?n 1 (upper-sum-holes A)
  proof
    fix y
    assume y-in: y ∈ sumset A A
    then obtain a b where a-in: a ∈ A and b-in: b ∈ A and y-eq: y = a + b
      by (rule sumsetE)
    have a-bounds: 0 ≤ a a ≤ ?n
      using subset a-in by auto
    have b-bounds: 0 ≤ b b ≤ ?n
      using subset b-in by auto
    show y ∈
        A ∪ affine-image-int ?n 1 A ∪
        lower-sum-holes A ∪
        affine-image-int ?n 1 (upper-sum-holes A)
```

```
  proof (cases y ≤ ?n)
    case True
    show ?thesis
    proof (cases y ∈ A)
      case True
      then show ?thesis
        by simp
    next
      case False
      have y ∈ interval-holes A
        using y-eq a-bounds b-bounds True False by (auto simp: interval-holes-def)
      then have y ∈ lower-sum-holes A
        using y-in by (simp add: lower-sum-holes-def)
      then show ?thesis
        by simp
    qed
  next
    case False
    let ?x = y − ?n
    have y-split: y = ?n + ?x
      by simp
    have x-pos: 0 < ?x
      using False by simp
    have x-le: ?x ≤ ?n
      using y-eq a-bounds b-bounds by linarith
    show ?thesis
    proof (cases ?x ∈ A)
      case True
      have y ∈ affine-image-int ?n 1 A
        unfolding affine-image-int-def
      proof (rule image-eqI[where x = ?x])
        show y = ?n + 1 ∗ ?x
          using y-split by simp
        show ?x ∈ A
          by (rule True)
      qed
      then show ?thesis
        by simp
    next
      case False
      have x-hole: ?x ∈ interval-holes A
        using x-pos x-le False by (auto simp: interval-holes-def)
      have x-upper: ?x ∈ upper-sum-holes A
        using x-hole y-in by (simp add: upper-sum-holes-def)
      have y ∈ affine-image-int ?n 1 (upper-sum-holes A)
        unfolding affine-image-int-def
      proof (rule image-eqI[where x = ?x])
        show y = ?n + 1 ∗ ?x
          using y-split by simp
        show ?x ∈ upper-sum-holes A
          by (rule x-upper)
      qed
      then show ?thesis
        by simp
    qed
  qed
qed
show A ∪ affine-image-int ?n 1 A ∪
    lower-sum-holes A ∪
```

```
    affine-image-int ?n 1 (upper-sum-holes A) ⊆ sumset A A
  proof
    fix y
    assume y-in: y ∈ A ∪ affine-image-int ?n 1 A ∪
      lower-sum-holes A ∪
      affine-image-int ?n 1 (upper-sum-holes A)
    then show y ∈ sumset A A
    proof
      assume y-base-or-lower:
        y ∈ A ∪ affine-image-int ?n 1 A ∪ lower-sum-holes A
      then show ?thesis
      proof
        assume y-base: y ∈ A ∪ affine-image-int ?n 1 A
        then show ?thesis
        proof
          assume yA: y ∈ A
          have 0 + y ∈ sumset A A
            using zero yA by (rule sumsetI)
          then show ?thesis
            by simp
        next
          assume y ∈ affine-image-int ?n 1 A
          then obtain a where a-in: a ∈ A and y-eq: y = ?n + a
            by (auto simp: affine-image-int-def)
          have ?n + a ∈ sumset A A
            using top a-in by (rule sumsetI)
          then show ?thesis
            using y-eq by simp
        qed
      next
        assume y ∈ lower-sum-holes A
        then show ?thesis
          by (simp add: lower-sum-holes-def)
      qed
    next
      assume y ∈ affine-image-int ?n 1 (upper-sum-holes A)
      then obtain x where x-upper: x ∈ upper-sum-holes A and y-eq: y = ?n + x
        by (auto simp: affine-image-int-def)
      then have ?n + x ∈ sumset A A
        by (simp add: upper-sum-holes-def)
      then show ?thesis
        using y-eq by simp
    qed
  qed
qed

lemma normalized-sumset-card-eq-with-holes:
  fixes A :: int set
  assumes fin: finite A
  assumes zero: 0 ∈ A
  assumes nonneg: ⋀x. x ∈ A ⟹ 0 ≤ x
  shows card (sumset A A) =
    2 * card A - 1 + card (lower-sum-holes A) + card (upper-sum-holes A)
proof -
  let ?B = A ∪ affine-image-int (Max A) 1 A
  let ?L = lower-sum-holes A
  let ?U = affine-image-int (Max A) 1 (upper-sum-holes A)
  let ?S = ?B ∪ ?L ∪ ?U
  have finB: finite ?B
```

```
    by (rule finite-UnI[OF fin finite-affine-image-int[OF fin]])
  have finL: finite ?L
    by simp
  have finU: finite ?U
    by (rule finite-affine-image-int) simp
  have cardB: card ?B = 2 * card A - 1
    by (rule normalized-endpoint-union-card[OF fin zero nonneg])
  have cardU: card ?U = card (upper-sum-holes A)
    by (rule card-affine-image-int) simp-all
  have BL-disj: ?B ∩ ?L = {}
    by (rule lower-sum-holes-disjoint-endpoint-union[OF fin zero nonneg])
  have BU-disj: ?B ∩ ?U = {}
    by (rule upper-sum-holes-image-disjoint-endpoint-union[OF fin zero nonneg])
  have LU-disj: ?L ∩ ?U = {}
    by (rule lower-sum-holes-disjoint-upper-sum-holes-image[OF fin zero nonneg])
  have BL-card: card (?B ∪ ?L) = card ?B + card ?L
    by (rule card-Un-disjoint[OF finB finL BL-disj])
  have BL-U-disj: (?B ∪ ?L) ∩ ?U = {}
    using BU-disj LU-disj by auto
  have card (sumset A A) = card ?S
    by (simp add: normalized-sumset-eq-endpoint-union-with-holes[OF fin zero nonneg])
  also have ... = card (?B ∪ ?L) + card ?U
    by (rule card-Un-disjoint[OF finite-UnI[OF finB finL] finU BL-U-disj])
  also have ... = card ?B + card ?L + card ?U
    using BL-card by simp
  also have ... =
      2 * card A - 1 + card (lower-sum-holes A) + card (upper-sum-holes A)
    using cardB cardU by simp
  finally show ?thesis .
qed

lemma normalized-small-doubling-hole-contribution-upper:
  fixes A :: int set
  assumes fin: finite A
  assumes card-ge: 3 ≤ card A
  assumes zero: 0 ∈ A
  assumes nonneg: ⋀x. x ∈ A ⟹ 0 ≤ x
  assumes small-doubling: card (sumset A A) ≤ 3 * card A - 4
  shows card (lower-sum-holes A) + card (upper-sum-holes A) ≤ card A - 3
proof -
  have card-eq:
    card (sumset A A) =
      2 * card A - 1 + card (lower-sum-holes A) + card (upper-sum-holes A)
    by (rule normalized-sumset-card-eq-with-holes[OF fin zero nonneg])
  show ?thesis
    using card-eq small-doubling card-ge by linarith
qed

lemma normalized-max-bound-from-hole-contribution:
  fixes A :: int set
  assumes fin: finite A
  assumes zero: 0 ∈ A
  assumes nonneg: ⋀x. x ∈ A ⟹ 0 ≤ x
  assumes hole-cover:
    card (interval-holes A) ≤ card (lower-sum-holes A) + card (upper-sum-holes A)
  shows nat (Max A + 1) ≤ card (sumset A A) - card A + 1
proof -
  let ?k = card A
  let ?s = card (sumset A A)
```

```
  let ?n = nat (Max A + 1)
  let ?h = card (interval-holes A)
  let ?l = card (lower-sum-holes A)
  let ?u = card (upper-sum-holes A)
  have lower: 2 * ?k - 1 + ?l + ?u ≤ ?s
    by (rule normalized-sumset-lower-bound-with-holes[OF fin zero nonneg])
  have h-bound: 2 * ?k - 1 + ?h ≤ ?s
    using lower hole-cover by linarith
  have subset: A ⊆ {0..Max A}
    by (rule normalized-subset-interval[OF fin zero nonneg])
  have max-nonneg: 0 ≤ Max A
    using fin zero by simp
  have k-le-n: ?k ≤ ?n
  proof -
    have ?k ≤ card {0..Max A}
      by (rule card-mono[OF - subset]) simp
    also have ... = ?n
      using max-nonneg by simp
    finally show ?thesis .
  qed
  have h-eq: ?h = ?n - ?k
    by (rule card-interval-holes[OF fin zero nonneg])
  have k-pos: 0 < ?k
    using fin zero by (simp add: card-gt-0-iff, blast)
  have n-eq: ?n = ?h + ?k
    using h-eq k-le-n by linarith
  have n-plus: ?n + ?k - 1 = 2 * ?k - 1 + ?h
    using n-eq k-pos by linarith
  have ?n + ?k - 1 ≤ ?s
    using h-bound n-plus by simp
  moreover note k-pos
  ultimately show ?thesis
    by linarith
qed
```

4.5 Modular shadows of integer sumsets

```
definition mod-image-int :: int ⇒ int set ⇒ int set where
  mod-image-int n A = (λx. x mod n) ‘ A

definition mod-sumset-int :: int ⇒ int set ⇒ int set ⇒ int set where
  mod-sumset-int n A B = mod-image-int n (sumset A B)

definition mod-translate-int :: int ⇒ int ⇒ int set ⇒ int set where
  mod-translate-int n a H = (λh. (a + h) mod n) ‘ H

definition mod-fiber-int :: int ⇒ int set ⇒ int ⇒ int set where
  mod-fiber-int n S r = {s ∈ S. s mod n = r}

lemma mod-image-int-iff:
  r ∈ mod-image-int n A ⟷ (∃ a∈A. r = a mod n)
  by (auto simp: mod-image-int-def)

lemma finite-mod-image-int [intro]:
  assumes finite A
  shows finite (mod-image-int n A)
  using assms by (simp add: mod-image-int-def)

lemma mod-sumset-int-iff:
```

```
  r ∈ mod-sumset-int n A B ⟷ (∃ a∈A. ∃ b∈B. r = (a + b) mod n)
  by (auto simp: mod-sumset-int-def mod-image-int-def sumset-def)

lemma finite-mod-sumset-int [intro]:
  assumes finite A finite B
  shows finite (mod-sumset-int n A B)
  unfolding mod-sumset-int-def
  by (rule finite-mod-image-int) (rule finite-sumset[OF assms])

lemma mod-translate-int-iff:
  r ∈ mod-translate-int n a H ⟷ (∃ h∈H. r = (a + h) mod n)
  by (auto simp: mod-translate-int-def)

lemma finite-mod-translate-int [intro]:
  assumes finite H
  shows finite (mod-translate-int n a H)
  using assms by (simp add: mod-translate-int-def)

lemma mod-translate-int-subset-residues:
  assumes n-pos: 0 < n
  shows mod-translate-int n a H ⊆ {0..n − 1}
  using n-pos by (auto simp: mod-translate-int-def pos-mod-bound)

lemma mod-add-translate-inj-on-residues:
  fixes a n :: int
  assumes n-pos: 0 < n
  shows inj-on (λh. (a + h) mod n) {0..n − 1}
proof (rule inj-onI)
  fix x y
  assume x-in: x ∈ {0..n − 1}
  assume y-in: y ∈ {0..n − 1}
  assume eq: (a + x) mod n = (a + y) mod n
  have x-bounds: 0 ≤ x x < n
    using x-in by auto
  have y-bounds: 0 ≤ y y < n
    using y-in by auto
  have dvd-xy: n dvd x − y
    using eq by (simp add: mod-eq-dvd-iff algebra-simps)
  then obtain q where q-eq: x − y = n ∗ q
    by (auto elim: dvdE)
  have diff-lower: − n < x − y
    using x-bounds y-bounds by linarith
  have diff-upper: x − y < n
    using x-bounds y-bounds by linarith
  have q = 0
  proof (rule ccontr)
    assume q ≠ 0
    then consider 1 ≤ q | q ≤ −1
      by linarith
    then show False
    proof cases
      case 1
      then have n ≤ n ∗ q
      proof −
        have 1 ∗ n ≤ q ∗ n
          using 1 n-pos by (intro mult-right-mono) simp-all
        then show ?thesis
          by (simp add: mult.commute)
      qed
```

```
      then show False
        using q-eq diff-upper by linarith
    next
      case 2
      then have n * q ≤ - n
      proof -
        have q * n ≤ (-1) * n
          using 2 n-pos by (intro mult-right-mono) simp-all
        then show ?thesis
          by (simp add: mult.commute)
      qed
      then show False
        using q-eq diff-lower by linarith
    qed
  qed
  then show x = y
    using q-eq by simp
qed

lemma card-mod-translate-int-eq:
  fixes H :: int set
  assumes n-pos: 0 < n
  assumes H-sub: H ⊆ {0..n - 1}
  shows card (mod-translate-int n a H) = card H
proof -
  have finH: finite H
    using H-sub by (rule finite-subset) simp
  have inj-res: inj-on (λh. (a + h) mod n) {0..n - 1}
    by (rule mod-add-translate-inj-on-residues[OF n-pos])
  have inj-H: inj-on (λh. (a + h) mod n) H
    by (rule inj-on-subset[OF inj-res H-sub])
  show ?thesis
    by (simp add: mod-translate-int-def card-image finH inj-H)
qed

lemma sum-coset-lower-upper-inter-card:
  fixes A H :: int set
  assumes n-pos: 0 < n
  assumes finA: finite A
  assumes max-eq: Max A = n
  assumes H-sub: H ⊆ {0..n - 1}
  assumes zero-H: 0 ∈ H
  assumes add-closed: ⋀x y. x ∈ H ⟹ y ∈ H ⟹ (x + y) mod n ∈ H
  assumes b-in: b ∈ A and b-bounds: 0 ≤ b b < n
  assumes c-in: c ∈ A and c-bounds: 0 ≤ c c < n
  defines R ≡ mod-translate-int n b H
  defines S ≡ mod-translate-int n c H
  defines K ≡ mod-translate-int n ((b + c) mod n) H
  assumes K-disj: K ∩ A = {}
  shows card H ≤
    1 + card ((lower-sum-holes A ∩ upper-sum-holes A) ∩ K) +
      card (R - A) + card (S - A)
proof -
  let ?X = A ∩ R
  let ?Y = A ∩ S
  let ?T = sumset ?X ?Y
  let ?Low = ?T ∩ {0..n - 1}
  let ?High = ?T ∩ {n..2 * n - 1}
  let ?Up = (λt. t - n) ` ?High
```

let *?I* = (*lower-sum-holes A* ∩ *upper-sum-holes A*) ∩ *K*
have *finH*: *finite H*
using *H-sub* **by** (*rule finite-subset*) *simp*
have *R-sub*: *R* ⊆ {*0..n* − *1*}
unfolding *R-def* **by** (*rule mod-translate-int-subset-residues*[*OF n-pos*])
have *S-sub*: *S* ⊆ {*0..n* − *1*}
unfolding *S-def* **by** (*rule mod-translate-int-subset-residues*[*OF n-pos*])
have *K-sub*: *K* ⊆ {*0..n* − *1*}
unfolding *K-def* **by** (*rule mod-translate-int-subset-residues*[*OF n-pos*])
have *finR*: *finite R*
using *R-sub* **by** (*rule finite-subset*) *simp*
have *finS*: *finite S*
using *S-sub* **by** (*rule finite-subset*) *simp*
have *finK*: *finite K*
using *K-sub* **by** (*rule finite-subset*) *simp*
have *cardR*: *card R* = *card H*
unfolding *R-def* **by** (*rule card-mod-translate-int-eq*[*OF n-pos H-sub*])
have *cardS*: *card S* = *card H*
unfolding *S-def* **by** (*rule card-mod-translate-int-eq*[*OF n-pos H-sub*])
have *cardK*: *card K* = *card H*
unfolding *K-def* **by** (*rule card-mod-translate-int-eq*[*OF n-pos H-sub*])
have *b-R*: *b* ∈ *R*
unfolding *R-def mod-translate-int-def*
proof (*rule image-eqI*)
show *b* = (*b* + *0*) *mod n*
using *b-bounds n-pos* **by** *simp*
show *0* ∈ *H*
by (*rule zero-H*)
qed
have *c-S*: *c* ∈ *S*
unfolding *S-def mod-translate-int-def*
proof (*rule image-eqI*)
show *c* = (*c* + *0*) *mod n*
using *c-bounds n-pos* **by** *simp*
show *0* ∈ *H*
by (*rule zero-H*)
qed
have *X-nonempty*: *?X* ≠ {}
using *b-in b-R* **by** *auto*
have *Y-nonempty*: *?Y* ≠ {}
using *c-in c-S* **by** *auto*
have *finX*: *finite ?X*
using *finA* **by** *simp*
have *finY*: *finite ?Y*
using *finA* **by** *simp*
have *sum-residue-in-K*: (*x* + *y*) *mod n* ∈ *K*
if *x-in*: *x* ∈ *?X* **and** *y-in*: *y* ∈ *?Y* **for** *x y*
proof −
from *x-in* **obtain** *h* **where** *h-in*: *h* ∈ *H* **and** *x-eq*: *x* = (*b* + *h*) *mod n*
by (*auto simp*: *R-def mod-translate-int-def*)
from *y-in* **obtain** *k* **where** *k-in*: *k* ∈ *H* **and** *y-eq*: *y* = (*c* + *k*) *mod n*
by (*auto simp*: *S-def mod-translate-int-def*)
have *hk-in*: (*h* + *k*) *mod n* ∈ *H*
by (*rule add-closed*[*OF h-in k-in*])
have (*x* + *y*) *mod n* = (*b* + *c* + ((*h* + *k*) *mod n*)) *mod n*
proof −
have (*x* + *y*) *mod n* = ((*b* + *h*) + (*c* + *k*)) *mod n*
using *x-eq y-eq* **by** (*simp add*: *mod-simps*)
also have ... = (*b* + *c* + (*h* + *k*)) *mod n*

```
    by (simp add: algebra-simps)
  also have ... = (b + c + ((h + k) mod n)) mod n
    by (simp add: mod-simps)
  finally show ?thesis .
 qed
 then show ?thesis
  using hk-in by (auto simp: K-def mod-translate-int-def mod-simps algebra-simps)
qed
have T-bounds: 0 ≤ t ∧ t ≤ 2 * n - 2 if t-in: t ∈ ?T for t
proof -
 from t-in obtain x y where x-in: x ∈ ?X and y-in: y ∈ ?Y and t-eq: t = x + y
  by (rule sumsetE)
 have x-bounds: 0 ≤ x x ≤ n - 1
  using R-sub x-in by auto
 have y-bounds: 0 ≤ y y ≤ n - 1
  using S-sub y-in by auto
 show ?thesis
  using x-bounds y-bounds t-eq by linarith
qed
have T-mod-K: t mod n ∈ K if t-in: t ∈ ?T for t
proof -
 from t-in obtain x y where x-in: x ∈ ?X and y-in: y ∈ ?Y and t-eq: t = x + y
  by (rule sumsetE)
 show ?thesis
  using sum-residue-in-K[OF x-in y-in] t-eq by simp
qed
have low-sub: ?Low ⊆ lower-sum-holes A ∩ K
proof
 fix x
 assume x-in: x ∈ ?Low
 then have x-T: x ∈ ?T and x-bounds: 0 ≤ x x < n
  by auto
 have x-K: x ∈ K
  using T-mod-K[OF x-T] x-bounds n-pos by simp
 have x-not-A: x ∉ A
  using K-disj x-K by auto
 have x-interval: x ∈ interval-holes A
  using x-bounds x-not-A by (auto simp: interval-holes-def max-eq)
 have x-sum: x ∈ sumset A A
  using x-T by (auto simp: sumset-def)
 show x ∈ lower-sum-holes A ∩ K
  using x-interval x-sum x-K by (auto simp: lower-sum-holes-def)
qed
have up-sub: ?Up ⊆ upper-sum-holes A ∩ K
proof
 fix x
 assume x-in: x ∈ ?Up
 then obtain t where t-high: t ∈ ?High and x-eq: x = t - n
  by auto
 then have t-T: t ∈ ?T and t-bounds: n ≤ t t < 2 * n
  by auto
 have x-bounds: 0 ≤ x x < n
  using x-eq t-bounds by linarith+
 have t-mod: t mod n = x
 proof -
  have t = n + x
   using x-eq by simp
  then show ?thesis
   using x-bounds n-pos by simp
```

```
  qed
  have x-K: x ∈ K
    using T-mod-K[OF t-T] t-mod by simp
  have x-not-A: x ∉ A
    using K-disj x-K by auto
  have x-interval: x ∈ interval-holes A
    using x-bounds x-not-A by (auto simp: interval-holes-def max-eq)
  have n + x ∈ sumset A A
    using t-T x-eq by (auto simp: sumset-def)
  then show x ∈ upper-sum-holes A ∩ K
    using x-interval x-K by (auto simp: upper-sum-holes-def max-eq)
qed
have finT: finite ?T
  by (rule finite-sumset[OF finX finY])
have T-split: ?T = ?Low ∪ ?High
proof
  show ?T ⊆ ?Low ∪ ?High
  proof
    fix t
    assume t-in: t ∈ ?T
    then have bounds: 0 ≤ t t ≤ 2 * n - 2
      using T-bounds by auto
    show t ∈ ?Low ∪ ?High
    proof (cases t < n)
      case True
      with t-in bounds show ?thesis
        by auto
    next
      case False
      with t-in bounds show ?thesis
        by auto
    qed
  qed
  show ?Low ∪ ?High ⊆ ?T
    by auto
qed
have disj-low-high: ?Low ∩ ?High = {}
  using n-pos by auto
have finLow: finite ?Low
  using finT by simp
have finHigh: finite ?High
  using finT by simp
have card-T-split: card ?T = card ?Low + card ?High
proof -
  have card ?T = card (?Low ∪ ?High)
    using T-split by simp
  also have ... = card ?Low + card ?High
    by (rule card-Un-disjoint[OF finLow finHigh disj-low-high])
  finally show ?thesis .
qed
have inj-shift: inj-on (λt. t - n) ?High
  by (rule inj-onI) simp
have card-Up: card ?Up = card ?High
  by (simp add: card-image finHigh inj-shift)
have card-T-low-up: card ?T = card ?Low + card ?Up
  using card-T-split card-Up by simp
have low-union-up-sub-K: ?Low ∪ ?Up ⊆ K
  using low-sub up-sub by auto
have low-inter-up-sub-I: ?Low ∩ ?Up ⊆ ?I
```

```
    using low-sub up-sub by auto
  have finUp: finite ?Up
    using finHigh by simp
  have card-low-up-le: card ?Low + card ?Up ≤ card K + card ?I
  proof −
    have card-union-inter:
        card ?Low + card ?Up = card (?Low ∪ ?Up) + card (?Low ∩ ?Up)
      by (rule card-Un-Int[OF finLow finUp])
    have card (?Low ∪ ?Up) ≤ card K
      by (rule card-mono[OF finK low-union-up-sub-K])
    moreover have card (?Low ∩ ?Up) ≤ card ?I
      by (rule card-mono[OF - low-inter-up-sub-I]) simp
    ultimately show ?thesis
      using card-union-inter by linarith
  qed
  have card-T-le: card ?T ≤ card H + card ?I
    using card-T-low-up card-low-up-le cardK by simp
  have sum-lower: card ?X + card ?Y − 1 ≤ card ?T
    by (rule card-sumset-ge-card-add-card-minus-one[OF finX finY X-nonempty Y-nonempty])
  have R-decomp: R = ?X ∪ (R − A)
    by auto
  have S-decomp: S = ?Y ∪ (S − A)
    by auto
  have R-disj: ?X ∩ (R − A) = {}
    by auto
  have S-disj: ?Y ∩ (S − A) = {}
    by auto
  have finRdiff: finite (R − A)
    using finR by simp
  have finSdiff: finite (S − A)
    using finS by simp
  have card-R-decomp: card R = card ?X + card (R − A)
  proof −
    have card R = card (?X ∪ (R − A))
      using R-decomp by simp
    also have ... = card ?X + card (R − A)
      by (rule card-Un-disjoint[OF finX finRdiff R-disj])
    finally show ?thesis .
  qed
  have card-S-decomp: card S = card ?Y + card (S − A)
  proof −
    have card S = card (?Y ∪ (S − A))
      using S-decomp by simp
    also have ... = card ?Y + card (S − A)
      by (rule card-Un-disjoint[OF finY finSdiff S-disj])
    finally show ?thesis .
  qed
  show ?thesis
    using sum-lower card-T-le cardR cardS card-R-decomp card-S-decomp by linarith
qed

lemma mod-fiber-int-subset:
  mod-fiber-int n S r ⊆ S
  by (auto simp: mod-fiber-int-def)

lemma finite-mod-fiber-int [intro]:
  assumes finite S
  shows finite (mod-fiber-int n S r)
  using finite-subset[OF mod-fiber-int-subset assms] .
```

interpretation *Zmod*: *additive-abelian-group* $\{0..int\ ((p :: nat) - 1)\}$
$(\lambda x\ y.\ (x + y)\ mod\ int\ p)\ 0::int$
using *additive-abelian-group-def residue-group*[*of p*] **by** *fastforce*

lemma *zmod-sumset-eq-mod-sumset-int*:
fixes $A\ B :: int\ set$
assumes *p-pos*: $0 < p$
assumes *A-sub*: $A \subseteq \{0..int\ p - 1\}$
assumes *B-sub*: $B \subseteq \{0..int\ p - 1\}$
shows $Zmod.sumset\ p\ A\ B = mod\text{-}sumset\text{-}int\ (int\ p)\ A\ B$
proof
have *zgroup*: *additive-abelian-group* $\{0..int\ (p - 1)\}$
$(\lambda x\ y.\ (x + y)\ mod\ int\ p)\ (0::int)$
using *additive-abelian-group-def residue-group*[*of p*] **by** *fastforce*
have *A-carrier*: $A \subseteq \{0..int\ (p - 1)\}$
using *p-pos A-sub* **by** *auto*
have *B-carrier*: $B \subseteq \{0..int\ (p - 1)\}$
using *p-pos B-sub* **by** *auto*
show $Zmod.sumset\ p\ A\ B \subseteq mod\text{-}sumset\text{-}int\ (int\ p)\ A\ B$
proof
fix x
assume $x \in Zmod.sumset\ p\ A\ B$
then have $x \in \{c.\ \exists a \in A \cap \{0..int\ (p - 1)\}.$
$\exists b \in B \cap \{0..int\ (p - 1)\}.\ c = (a + b)\ mod\ int\ p\}$
using *additive-abelian-group.sumset-eq*[*OF zgroup, of A B*] **by** *auto*
then obtain $a\ b$ **where** *a-in*: $a \in A$ **and** *b-in*: $b \in B$
and *x-eq*: $x = (a + b)\ mod\ int\ p$
by *auto*
have $a + b \in sumset\ A\ B$
using *a-in b-in* **by** (*rule sumsetI*)
then show $x \in mod\text{-}sumset\text{-}int\ (int\ p)\ A\ B$
using *x-eq* **by** (*auto simp*: *mod-sumset-int-def mod-image-int-def*)
qed
show $mod\text{-}sumset\text{-}int\ (int\ p)\ A\ B \subseteq Zmod.sumset\ p\ A\ B$
proof
fix x
assume $x \in mod\text{-}sumset\text{-}int\ (int\ p)\ A\ B$
then obtain $a\ b$ **where** *a-in*: $a \in A$ **and** *b-in*: $b \in B$
and *x-eq*: $x = (a + b)\ mod\ int\ p$
by (*auto simp*: *mod-sumset-int-iff*)
have *a-carrier*: $a \in \{0..int\ (p - 1)\}$
using *A-carrier a-in* **by** *auto*
have *b-carrier*: $b \in \{0..int\ (p - 1)\}$
using *B-carrier b-in* **by** *auto*
have $x \in \{c.\ \exists a \in A \cap \{0..int\ (p - 1)\}.$
$\exists b \in B \cap \{0..int\ (p - 1)\}.\ c = (a + b)\ mod\ int\ p\}$
using *a-in a-carrier b-in b-carrier x-eq* **by** *blast*
show $x \in Zmod.sumset\ p\ A\ B$
using ‹$x \in \{c.\ \exists a \in A \cap \{0..int\ (p - 1)\}.$
$\exists b \in B \cap \{0..int\ (p - 1)\}.\ c = (a + b)\ mod\ int\ p\}$›
additive-abelian-group.sumset-eq[*OF zgroup, of A B*]
by *auto*
qed
qed

lemma *zmod-kneser-self*:
fixes $B :: int\ set$
assumes *p-pos*: $0 < p$

```
  assumes B-sub: B ⊆ {0..int p − 1}
  assumes fin: finite B
  assumes nonempty: B ≠ {}
  defines C ≡ Zmod.sumset p B B
  defines H ≡ Zmod.stabilizer p C
  shows card C ≥ 2 ∗ card (Zmod.sumset p B H) − card H
proof −
  have B-carrier: B ⊆ {0..int (p − 1)}
    using p-pos B-sub by auto
  have card (Zmod.sumset p B H) + card (Zmod.sumset p B H) − card H ≤ card C
    unfolding C-def H-def
    by (rule Zmod.Kneser[OF B-carrier B-carrier fin fin nonempty nonempty])
  then show ?thesis
    by simp
qed

lemma zmod-sumset-iff:
  x ∈ Zmod.sumset p A B ⟷
    (∃ a∈A ∩ {0..int (p − 1)}.
      ∃ b∈B ∩ {0..int (p − 1)}. x = (a + b) mod int p)
proof −
  have zgroup: additive-abelian-group {0..int (p − 1)}
      (λx y. (x + y) mod int p) (0::int)
    using additive-abelian-group-def residue-group[of p] by fastforce
  show ?thesis
    using additive-abelian-group.sumset-eq[OF zgroup, of A B] by auto
qed

lemma zmod-singleton-sumset-eq-mod-translate-int:
  assumes p-pos: 0 < p
  assumes a-carrier: a ∈ {0..int p − 1}
  assumes H-sub: H ⊆ {0..int p − 1}
  shows Zmod.sumset p {a} H = mod-translate-int (int p) a H
proof
  have carrier-eq: {0..int (p − 1)} = {0..int p − 1}
    using p-pos by auto
  show Zmod.sumset p {a} H ⊆ mod-translate-int (int p) a H
  proof
    fix x
    assume x-in: x ∈ Zmod.sumset p {a} H
    then obtain h where h-in: h ∈ H and x-eq: x = (a + h) mod int p
      using zmod-sumset-iff[of x p {a} H] by blast
    show x ∈ mod-translate-int (int p) a H
      using h-in x-eq by (auto simp: mod-translate-int-def)
  qed
  show mod-translate-int (int p) a H ⊆ Zmod.sumset p {a} H
  proof
    fix x
    assume x-in: x ∈ mod-translate-int (int p) a H
    then obtain h where h-in: h ∈ H and x-eq: x = (a + h) mod int p
      by (auto simp: mod-translate-int-def)
    have a-carrier': a ∈ {0..int (p − 1)}
      using a-carrier carrier-eq by simp
    have h-carrier: h ∈ {0..int (p − 1)}
      using H-sub h-in carrier-eq by auto
    have (a + h) mod int p ∈ Zmod.sumset p {a} H
      by (rule Zmod.sumset.sumsetI) (use a-carrier' h-carrier h-in in auto)
    then show x ∈ Zmod.sumset p {a} H
      using x-eq by simp
```

```
  qed
qed

lemma zmod-self-sumset-contains-set:
  assumes p-pos: 0 < p
  assumes B-sub: B ⊆ {0..int p − 1}
  assumes zero-B: 0 ∈ B
  shows B ⊆ Zmod.sumset p B B
proof
  fix b
  assume b-in: b ∈ B
  have b-carrier: b ∈ {0..int (p − 1)}
    using B-sub b-in p-pos by auto
  have zero-carrier: 0 ∈ {0..int (p − 1)}
    using p-pos by auto
  have (b + 0) mod int p ∈ Zmod.sumset p B B
    by (rule Zmod.sumset.sumsetI[OF b-in b-carrier zero-B zero-carrier])
  then show b ∈ Zmod.sumset p B B
    using b-carrier p-pos by simp
qed

lemma zmod-sumset-stabilizer-subset:
  assumes p-pos: 0 < p
  assumes A-sub-C: A ⊆ C
  assumes C-sub: C ⊆ {0..int p − 1}
  shows Zmod.sumset p A (Zmod.stabilizer p C) ⊆ C
proof
  have carrier-eq: {0..int (p − 1)} = {0..int p − 1}
    using p-pos by auto
  fix x
  assume x-in: x ∈ Zmod.sumset p A (Zmod.stabilizer p C)
  then obtain a h where a-in: a ∈ A and a-carrier: a ∈ {0..int (p − 1)}
      and h-in: h ∈ Zmod.stabilizer p C and h-carrier: h ∈ {0..int (p − 1)}
      and x-eq: x = (a + h) mod int p
    using zmod-sumset-iff[of x p A Zmod.stabilizer p C] by blast
  have a-C: a ∈ C
    using A-sub-C a-in by auto
  have C-sub': C ⊆ {0..int (p − 1)}
    using C-sub carrier-eq by simp
  have coset-sub: Zmod.sumset p {a} (Zmod.stabilizer p C) ⊆ C
    by (rule Zmod.stabilizer-coset-subset[OF C-sub' a-C])
  have x ∈ Zmod.sumset p {a} (Zmod.stabilizer p C)
  proof −
    have (a + h) mod int p ∈ Zmod.sumset p {a} (Zmod.stabilizer p C)
      by (rule Zmod.sumset.sumsetI) (use a-carrier h-in h-carrier in auto)
    then show ?thesis
      using x-eq by simp
  qed
  then show x ∈ C
    using coset-sub by auto
qed

lemma zmod-obtain-sum-coset-disjoint-D:
  fixes B :: int set
  assumes p-pos: 0 < p
  assumes B-sub: B ⊆ {0..int p − 1}
  assumes finB: finite B
  assumes zero-B: 0 ∈ B
  defines C ≡ Zmod.sumset p B B
```

```
  defines H ≡ Zmod.stabilizer p C
  defines D ≡ Zmod.sumset p B H
  assumes not-subset: ¬ B ⊆ H
  obtains b c where b ∈ B c ∈ B Zmod.sumset p {((b + c) mod int p)} H ∩ D = {}
proof −
  have carrier-eq: {0..int (p − 1)} = {0..int p − 1}
    using p-pos by auto
  have exists-coset:
    ∃ b c. b ∈ B ∧ c ∈ B ∧ Zmod.sumset p {((b + c) mod int p)} H ∩ D = {}
  proof (rule ccontr)
    assume no-coset:
      ¬ (∃ b c. b ∈ B ∧ c ∈ B ∧
        Zmod.sumset p {((b + c) mod int p)} H ∩ D = {})
    then have all-meet:
      Zmod.sumset p {((b + c) mod int p)} H ∩ D ≠ {}
      if b ∈ B c ∈ B for b c
      using that by blast
    have C-sub: C ⊆ {0..int p − 1}
      unfolding C-def
      using Zmod.sumset-subset-carrier[of p B B] carrier-eq by auto
    have C-sub': C ⊆ {0..int (p − 1)}
      using C-sub carrier-eq by simp
    have H-sub: H ⊆ {0..int p − 1}
      unfolding H-def using Zmod.stabilizer-subset-group[of p C] carrier-eq by auto
    have zero-H: 0 ∈ H
      unfolding H-def by (rule Zmod.zero-mem-stabilizer)
    have B-sub-C: B ⊆ C
      unfolding C-def by (rule zmod-self-sumset-contains-set[OF p-pos B-sub zero-B])
    have D-sub-C: D ⊆ C
      unfolding D-def H-def
      by (rule zmod-sumset-stabilizer-subset[OF p-pos B-sub-C C-sub])
    have C-sub-D: C ⊆ D
    proof
    fix r
    assume r-in: r ∈ C
    then obtain b c where b-in: b ∈ B and c-in: c ∈ B
        and r-eq: r = (b + c) mod int p
      unfolding C-def using zmod-sumset-iff[of r p B B] by blast
    have r-carrier: r ∈ {0..int (p − 1)}
      using C-sub' r-in by auto
    have coset-meet: Zmod.sumset p {r} H ∩ D ≠ {}
      using all-meet[OF b-in c-in] r-eq by simp
    then obtain z where z-r: z ∈ Zmod.sumset p {r} H and z-D: z ∈ D
      by auto
    from z-D obtain b0 h0 where b0-in: b0 ∈ B and h0-in: h0 ∈ H
        and z-eq: z = (b0 + h0) mod int p
      unfolding D-def using zmod-sumset-iff[of z p B H] by blast
    have b0-carrier: b0 ∈ {0..int (p − 1)}
      using B-sub b0-in carrier-eq by auto
    have h0-carrier: h0 ∈ {0..int (p − 1)}
      using H-sub h0-in carrier-eq by auto
    have z-b0: z ∈ Zmod.sumset p {b0} H
    proof −
      have (b0 + h0) mod int p ∈ Zmod.sumset p {b0} H
        by (rule Zmod.sumset.sumsetI) (use b0-carrier h0-carrier h0-in in auto)
      then show ?thesis
        using z-eq by simp
    qed
    have b0-coset-sub-D: Zmod.sumset p {b0} H ⊆ D
```

```
    unfolding D-def
    by (rule Zmod.sumset-mono) (use b0-in in auto)
  have cosets-inter: Zmod.sumset p {r} H ∩ Zmod.sumset p {b0} H ≠ {}
    using z-r z-b0 by auto
  have cosets-eq: Zmod.sumset p {r} H = Zmod.sumset p {b0} H
    using Zmod.sumset-stabilizer-eq-iff[OF r-carrier b0-carrier, of C] cosets-inter
    by (simp add: H-def)
  have r-in-own: r ∈ Zmod.sumset p {r} H
  proof −
    have (r + 0) mod int p ∈ Zmod.sumset p {r} H
      by (rule Zmod.sumset.sumsetI) (use r-carrier zero-H p-pos in auto)
    then show ?thesis
      using r-carrier p-pos by simp
  qed
  show r ∈ D
    using r-in-own cosets-eq b0-coset-sub-D by auto
qed
have C-eq-D: C = D
  using C-sub-D D-sub-C by auto
have finC: finite C
  unfolding C-def by (rule Zmod.finite-sumset[OF finB finB])
have zero-C: 0 ∈ C
proof −
  have zero-carrier: 0 ∈ {0..int (p − 1)}
    using p-pos by auto
  have (0 + 0) mod int p ∈ Zmod.sumset p B B
    by (rule Zmod.sumset.sumsetI[OF zero-B zero-carrier zero-B zero-carrier])
  then show ?thesis
    by (simp add: C-def)
qed
have C-plus-C: Zmod.sumset p C C = C
proof
  show Zmod.sumset p C C ⊆ C
  proof
    fix z
    assume z-in: z ∈ Zmod.sumset p C C
    then obtain x y where x-in: x ∈ C and y-in: y ∈ C
        and x-carrier: x ∈ {0..int (p − 1)}
        and y-carrier: y ∈ {0..int (p − 1)}
        and z-eq: z = (x + y) mod int p
      using zmod-sumset-iff[of z p C C] by blast
    from x-in have x-D: x ∈ D
      using C-eq-D by simp
    from y-in have y-D: y ∈ D
      using C-eq-D by simp
    from x-D obtain b h where b-in: b ∈ B and h-in: h ∈ H
        and x-eq: x = (b + h) mod int p
      unfolding D-def using zmod-sumset-iff[of x p B H] by blast
    from y-D obtain c k where c-in: c ∈ B and k-in: k ∈ H
        and y-eq: y = (c + k) mod int p
      unfolding D-def using zmod-sumset-iff[of y p B H] by blast
    have b-carrier: b ∈ {0..int (p − 1)}
      using B-sub b-in carrier-eq by auto
    have c-carrier: c ∈ {0..int (p − 1)}
      using B-sub c-in carrier-eq by auto
    have h-carrier: h ∈ {0..int (p − 1)}
      using H-sub h-in carrier-eq by auto
    have k-carrier: k ∈ {0..int (p − 1)}
      using H-sub k-in carrier-eq by auto
```

```
    have bc-C: (b + c) mod int p ∈ C
      unfolding C-def by (rule Zmod.sumset.sumsetI[OF b-in b-carrier c-in c-carrier])
    have hk-H: (h + k) mod int p ∈ H
    proof −
      interpret Hsub: subgroup H {0..int (p − 1)}
          (λx y. (x + y) mod int p) 0::int
        unfolding H-def by (rule Zmod.stabilizer-is-subgroup)
      show ?thesis
        by (rule Hsub.sub-composition-closed[OF h-in k-in])
    qed
    have hk-carrier: (h + k) mod int p ∈ {0..int (p − 1)}
      using hk-H H-sub carrier-eq by auto
    have bc-carrier: (b + c) mod int p ∈ {0..int (p − 1)}
      using bc-C C-sub′ by auto
    have z-rewrite:
      z = (((b + c) mod int p) + ((h + k) mod int p)) mod int p
    proof −
      have z = ((b + h) + (c + k)) mod int p
        using z-eq x-eq y-eq by (simp add: mod-simps)
      also have ... = (b + c + (h + k)) mod int p
        by (simp add: algebra-simps)
      also have ... =
          (((b + c) mod int p) + ((h + k) mod int p)) mod int p
        by (simp add: mod-add-eq algebra-simps)
      finally show ?thesis .
    qed
    have z-in-CH: z ∈ Zmod.sumset p C H
    proof −
      have (((b + c) mod int p) + ((h + k) mod int p)) mod int p
          ∈ Zmod.sumset p C H
        by (rule Zmod.sumset.sumsetI[OF bc-C bc-carrier hk-H hk-carrier])
      then show ?thesis
        using z-rewrite by simp
    qed
    have CH-sub-C: Zmod.sumset p C H ⊆ C
      unfolding H-def
      by (rule zmod-sumset-stabilizer-subset[OF p-pos subset-refl C-sub])
    show z ∈ C
      using CH-sub-C z-in-CH by auto
  qed
next
  show C ⊆ Zmod.sumset p C C
  proof
    fix x
    assume x-in: x ∈ C
    have x-carrier: x ∈ {0..int (p − 1)}
      using C-sub′ x-in by auto
    have zero-carrier: 0 ∈ {0..int (p − 1)}
      using p-pos by auto
    have (x + 0) mod int p ∈ Zmod.sumset p C C
      by (rule Zmod.sumset.sumsetI[OF x-in x-carrier zero-C zero-carrier])
    then show x ∈ Zmod.sumset p C C
      using x-carrier p-pos by simp
  qed
qed
have C-sub-H: C ⊆ H
  unfolding H-def
  by (rule Zmod.sumset-eq-sub-stabilizer[OF C-sub′ C-sub′ finC C-plus-C])
  have B ⊆ H
```

```
    using B-sub-C C-sub-H by auto
  show False
    using ‹B ⊆ H› not-subset by simp
 qed
 then obtain b c where b-in: b ∈ B and c-in: c ∈ B
    and disj: Zmod.sumset p {((b + c) mod int p)} H ∩ D = {}
  by blast
 show ?thesis
  by (rule that[OF b-in c-in disj])
qed

lemma mod-image-int-subset-residues:
 assumes n-pos: 0 < n
 shows mod-image-int n A ⊆ {0..n − 1}
 using n-pos by (auto simp: mod-image-int-def pos-mod-bound)

lemma mod-sumset-int-mod-image-self:
 assumes n-pos: 0 < n
 shows mod-sumset-int n (mod-image-int n A) (mod-image-int n A) =
  mod-sumset-int n A A
proof
 show mod-sumset-int n (mod-image-int n A) (mod-image-int n A) ⊆
    mod-sumset-int n A A
 proof
  fix r
  assume r ∈ mod-sumset-int n (mod-image-int n A) (mod-image-int n A)
  then obtain a b where a-in: a ∈ A and b-in: b ∈ A
    and r-eq: r = (a mod n + b mod n) mod n
    by (auto simp: mod-sumset-int-iff mod-image-int-def)
  have (a mod n + b mod n) mod n = (a + b) mod n
    by (simp add: mod-simps)
  then show r ∈ mod-sumset-int n A A
    using a-in b-in r-eq by (auto simp: mod-sumset-int-iff)
 qed
 show mod-sumset-int n A A ⊆
    mod-sumset-int n (mod-image-int n A) (mod-image-int n A)
 proof
  fix r
  assume r ∈ mod-sumset-int n A A
  then obtain a b where a-in: a ∈ A and b-in: b ∈ A
    and r-eq: r = (a + b) mod n
    by (auto simp: mod-sumset-int-iff)
  have a-mod-in: a mod n ∈ mod-image-int n A
    using a-in by (auto simp: mod-image-int-def)
  have b-mod-in: b mod n ∈ mod-image-int n A
    using b-in by (auto simp: mod-image-int-def)
  have r = (a mod n + b mod n) mod n
    using r-eq by (simp add: mod-simps)
  then show r ∈ mod-sumset-int n (mod-image-int n A) (mod-image-int n A)
    using a-mod-in b-mod-in by (auto simp: mod-sumset-int-iff)
 qed
qed

lemma zmod-sumset-trivial-stabilizer-card-ge:
 fixes B :: int set
 assumes p-pos: 0 < p
 assumes B-sub: B ⊆ {0..int p − 1}
 assumes fin: finite B
 assumes nonempty: B ≠ {}
```

```
  defines C ≡ Zmod.sumset p B B
  assumes trivial: Zmod.stabilizer p C = {0}
  shows 2 * card B - 1 ≤ card C
proof -
  have B-carrier: B ⊆ {0..int (p - 1)}
    using p-pos B-sub by auto
  have zero-carrier: {0::int} ⊆ {0..int (p - 1)}
    using p-pos by auto
  have kneser-raw:
    card (Zmod.sumset p B (Zmod.stabilizer p C)) +
      card (Zmod.sumset p B (Zmod.stabilizer p C)) -
      card (Zmod.stabilizer p C) ≤ card C
    unfolding C-def
    by (rule Zmod.Kneser[OF B-carrier B-carrier fin fin nonempty nonempty])
  have stabilizer-rewrite:
    card (Zmod.sumset p B (Zmod.stabilizer p C)) +
      card (Zmod.sumset p B (Zmod.stabilizer p C)) -
      card (Zmod.stabilizer p C) = 2 * card (Zmod.sumset p B {0}) - 1
    using trivial by simp
  have zero-sumset: Zmod.sumset p B {0} = B
  proof -
    have Zmod.sumset p B {0} = B ∩ {0..int (p - 1)}
      by (rule Zmod.sumset-D(1))
    also have ... = B
      using B-carrier by auto
    finally show ?thesis .
  qed
  have 2 * card B - 1 =
      card (Zmod.sumset p B (Zmod.stabilizer p C)) +
      card (Zmod.sumset p B (Zmod.stabilizer p C)) -
      card (Zmod.stabilizer p C)
  proof -
    have 2 * card (Zmod.sumset p B {0}) - 1 = 2 * card B - 1
      by (subst zero-sumset) simp
    with stabilizer-rewrite show ?thesis
      by simp
  qed
  with kneser-raw show ?thesis
    by simp
qed

lemma zmod-nontrivial-stabilizer-obtain-positive-period:
  fixes C :: int set
  assumes p-pos: 0 < p
  assumes nontrivial: Zmod.stabilizer p C ≠ {0}
  obtains h where h ∈ Zmod.stabilizer p C 0 < h h < int p
proof -
  have zero-in: 0 ∈ Zmod.stabilizer p C
    by (rule Zmod.zero-mem-stabilizer)
  from nontrivial obtain h where h-in: h ∈ Zmod.stabilizer p C and h-ne: h ≠ 0
    using zero-in by auto
  have h-carrier: h ∈ {0..int (p - 1)}
    using h-in Zmod.stabilizer-subset-group[of p C] by auto
  then have h-nonneg: 0 ≤ h and h-le: h ≤ int (p - 1)
    by auto
  have h-pos: 0 < h
    using h-nonneg h-ne by linarith
  have h-lt: h < int p
    using h-le p-pos by linarith
```

```
  show ?thesis
    by (rule that[OF h-in h-pos h-lt])
qed

lemma residue-add-closed-mult:
  fixes H :: int set
  assumes n-pos: 0 < n
  assumes zero: 0 ∈ H
  assumes add-closed: ⋀x y. x ∈ H ⟹ y ∈ H ⟹ (x + y) mod n ∈ H
  assumes h-in: h ∈ H
  shows (int m * h) mod n ∈ H
proof (induction m)
  case 0
  then show ?case
    using zero by simp
next
  case (Suc m)
  have prev: (int m * h) mod n ∈ H
    by (rule Suc.IH)
  have step: (((int m * h) mod n) + h) mod n ∈ H
    by (rule add-closed[OF prev h-in])
  have (((int m * h) mod n) + h) mod n = (int (Suc m) * h) mod n
    by (simp add: mod-simps algebra-simps)
  with step show ?case
    by simp
qed

lemma residue-add-closed-diff:
  fixes H :: int set
  assumes n-pos: 0 < n
  assumes zero: 0 ∈ H
  assumes add-closed: ⋀x y. x ∈ H ⟹ y ∈ H ⟹ (x + y) mod n ∈ H
  assumes x-in: x ∈ H
  assumes y-in: y ∈ H
  shows (x - y) mod n ∈ H
proof -
  have neg-y: (int (nat n - 1) * y) mod n ∈ H
    by (rule residue-add-closed-mult[OF n-pos zero add-closed y-in])
  have int-pred: int (nat n - 1) = n - 1
    using n-pos by simp
  have inH: ((x + ((int (nat n - 1) * y) mod n)) mod n) ∈ H
    by (rule add-closed[OF x-in neg-y])
  have eq: ((x + ((int (nat n - 1) * y) mod n)) mod n) = (x - y) mod n
  proof -
    have ((x + ((int (nat n - 1) * y) mod n)) mod n) =
        (x + ((n - 1) * y)) mod n
      using int-pred by (simp add: mod-simps)
    also have ... = ((x - y) + n * y) mod n
      by (simp add: algebra-simps)
    also have ... = (x - y) mod n
      by (simp add: mod-simps)
    finally show ?thesis .
  qed
  from inH show ?thesis
    using eq by simp
qed

lemma residue-add-closed-obtain-proper-divisor:
  fixes H :: int set
```

```
  assumes n-pos: 0 < n
  assumes H-sub: H ⊆ {0..n − 1}
  assumes zero: 0 ∈ H
  assumes add-closed: ⋀x y. x ∈ H ⟹ y ∈ H ⟹ (x + y) mod n ∈ H
  assumes nontrivial: H ≠ {0}
  assumes proper: H ≠ {0..n − 1}
  obtains d where 1 < d d dvd n ⋀h. h ∈ H ⟹ d dvd h
proof −
  let ?S = H − {0}
  have finH: finite H
    using H-sub by (rule finite-subset) simp
  have finS: finite ?S
    using finH by simp
  have nonemptyS: ?S ≠ {}
    using zero nontrivial by auto
  define d where d = Min ?S
  have d-inS: d ∈ ?S
    unfolding d-def by (rule Min-in[OF finS nonemptyS])
  then have d-in: d ∈ H and d-ne-zero: d ≠ 0
    by auto
  have d-bounds: 0 < d d < n
  proof −
    have d ∈ {0..n − 1}
      using H-sub d-in by auto
    then have 0 ≤ d d < n
      by auto
    with d-ne-zero show 0 < d d < n
      by linarith+
  qed
  have dvd-H: d dvd h if h-in: h ∈ H for h
  proof (cases h = 0)
    case True
    then show ?thesis
      by simp
  next
    case False
    have h-inS: h ∈ ?S
      using h-in False by simp
    have h-bounds: 0 ≤ h h < n
      using H-sub h-in by auto
    have d-le-h: d ≤ h
      using Min-le[OF finS h-inS] by (simp add: d-def)
    define q where q = h div d
    define r where r = h mod d
    have q-nonneg: 0 ≤ q
      using h-bounds d-bounds by (simp add: q-def pos-imp-zdiv-nonneg-iff)
    have r-bounds: 0 ≤ r r < d
      using d-bounds by (simp-all add: r-def)
    have div-eq: h = q * d + r
      by (simp add: q-def r-def)
    have qd-bounds: 0 ≤ q * d q * d < n
      using q-nonneg d-bounds h-bounds div-eq r-bounds by (simp, linarith)
    have qd-mod-in: (int (nat q) * d) mod n ∈ H
      by (rule residue-add-closed-mult[OF n-pos zero add-closed d-in])
    have qd-in: q * d ∈ H
      using qd-mod-in q-nonneg qd-bounds n-pos by simp
    have r-mod-in: (h − q * d) mod n ∈ H
      by (rule residue-add-closed-diff[OF n-pos zero add-closed h-in qd-in])
    have r-eq: h − q * d = r
```

```
    using div-eq by simp
  have r-in: r ∈ H
    using r-mod-in r-eq r-bounds d-bounds n-pos by simp
  have r = 0
  proof (rule ccontr)
    assume r ≠ 0
    then have r-inS: r ∈ ?S
      using r-in by simp
    have d ≤ r
      using Min-le[OF finS r-inS] by (simp add: d-def)
    with r-bounds show False
      by linarith
  qed
  then have h = q * d
    using div-eq by simp
  then show ?thesis
    unfolding dvd-def by (metis mult.commute)
qed
have d-dvd-n: d dvd n
proof −
  define q where q = n div d
  define r where r = n mod d
  have q-nonneg: 0 ≤ q
    using n-pos d-bounds by (simp add: q-def pos-imp-zdiv-nonneg-iff)
  have r-bounds: 0 ≤ r r < d
    using d-bounds by (simp-all add: r-def)
  have n-eq: n = q * d + r
    by (simp add: q-def r-def)
  show ?thesis
  proof (cases r = 0)
    case True
    then have n = q * d
      using n-eq by simp
    then show ?thesis
      unfolding dvd-def by (metis mult.commute)
  next
    case False
    then have r-pos: 0 < r
      using r-bounds by linarith
    have qd-bounds: 0 < q * d q * d < n
      using n-eq r-bounds r-pos d-bounds n-pos by linarith+
    have qd-mod-in: (int (nat q) * d) mod n ∈ H
      by (rule residue-add-closed-mult[OF n-pos zero add-closed d-in])
    have qd-in: q * d ∈ H
      using qd-mod-in q-nonneg qd-bounds n-pos by simp
    have r-mod-in: (0 − q * d) mod n ∈ H
      by (rule residue-add-closed-diff[OF n-pos zero add-closed zero qd-in])
    have neg-eq: (0 − q * d) mod n = r
    proof −
      have q * d = n − r
        using n-eq by simp
      then have (0 − q * d) mod n = (r − n) mod n
        by simp
      also have ... = r mod n
        by (simp add: mod-simps)
      also have ... = r
        using r-bounds d-bounds n-pos by simp
      finally show ?thesis .
    qed
```

```
    have r-in: r ∈ H
      using r-mod-in neg-eq by simp
    have r-inS: r ∈ ?S
      using r-in r-pos by auto
    have d ≤ r
      using Min-le[OF finS r-inS] by (simp add: d-def)
    with r-bounds show ?thesis
      by linarith
  qed
 qed
 have d-gt-one: 1 < d
 proof (rule ccontr)
  assume ¬ 1 < d
  then have d-eq-one: d = 1
    using d-bounds by linarith
  have carrier-sub: {0..n − 1} ⊆ H
  proof
    fix x
    assume x-in: x ∈ {0..n − 1}
    then have x-bounds: 0 ≤ x x < n
      by auto
    have (int (nat x) ∗ d) mod n ∈ H
      by (rule residue-add-closed-mult[OF n-pos zero add-closed d-in])
    then show x ∈ H
      using x-bounds d-eq-one n-pos by simp
  qed
  have H = {0..n − 1}
    using H-sub carrier-sub by auto
  with proper show False
    by simp
 qed
 show ?thesis
  by (rule that[OF d-gt-one d-dvd-n dvd-H])
qed

lemma residue-add-closed-min-step-eq:
 fixes H :: int set
 assumes n-pos: 0 < n
 assumes H-sub: H ⊆ {0..n − 1}
 assumes zero: 0 ∈ H
 assumes add-closed: ⋀x y. x ∈ H ⟹ y ∈ H ⟹ (x + y) mod n ∈ H
 assumes nontrivial: H ≠ {0}
 defines d ≡ Min (H − {0})
 shows 0 < d
  and d dvd n
  and H = {x ∈ {0..n − 1}. d dvd x}
proof −
 let ?S = H − {0}
 have finH: finite H
  using H-sub by (rule finite-subset) simp
 have finS: finite ?S
  using finH by simp
 have nonemptyS: ?S ≠ {}
  using zero nontrivial by auto
 have d-inS: d ∈ ?S
  unfolding d-def by (rule Min-in[OF finS nonemptyS])
 then have d-in: d ∈ H and d-ne-zero: d ≠ 0
  by auto
 have d-bounds: 0 < d d < n
```

proof −
have $d \in \{0..n - 1\}$
using *H-sub d-in* **by** *auto*
then have $0 \leq d\ d < n$
by *auto*
with *d-ne-zero* **show** $0 < d\ d < n$
by *linarith*+
qed
have *dvd-H*: $d\ dvd\ h$ **if** *h-in*: $h \in H$ **for** h
proof (*cases* $h = 0$)
case *True*
then show *?thesis*
by *simp*
next
case *False*
have *h-inS*: $h \in ?S$
using *h-in False* **by** *simp*
have *h-bounds*: $0 \leq h\ h < n$
using *H-sub h-in* **by** *auto*
define q **where** $q = h\ div\ d$
define r **where** $r = h\ mod\ d$
have *q-nonneg*: $0 \leq q$
using *h-bounds d-bounds* **by** (*simp add*: *q-def pos-imp-zdiv-nonneg-iff*)
have *r-bounds*: $0 \leq r\ r < d$
using *d-bounds* **by** (*simp-all add*: *r-def*)
have *div-eq*: $h = q * d + r$
by (*simp add*: *q-def r-def*)
have *qd-bounds*: $0 \leq q * d\ q * d < n$
using *q-nonneg d-bounds h-bounds div-eq r-bounds* **by** (*simp*, *linarith*)
have *qd-mod-in*: $(int\ (nat\ q) * d)\ mod\ n \in H$
by (*rule residue-add-closed-mult*[*OF n-pos zero add-closed d-in*])
have *qd-in*: $q * d \in H$
using *qd-mod-in q-nonneg qd-bounds n-pos* **by** *simp*
have *r-mod-in*: $(h - q * d)\ mod\ n \in H$
by (*rule residue-add-closed-diff*[*OF n-pos zero add-closed h-in qd-in*])
have *r-eq*: $h - q * d = r$
using *div-eq* **by** *simp*
have *r-in*: $r \in H$
using *r-mod-in r-eq r-bounds d-bounds n-pos* **by** *simp*
have $r = 0$
proof (*rule ccontr*)
assume $r \neq 0$
then have *r-inS*: $r \in ?S$
using *r-in* **by** *simp*
have $d \leq r$
using *Min-le*[*OF finS r-inS*] **by** (*simp add*: *d-def*)
with *r-bounds* **show** *False*
by *linarith*
qed
then have $h = q * d$
using *div-eq* **by** *simp*
then show *?thesis*
unfolding *dvd-def* **by** (*metis mult.commute*)
qed
have *d-dvd-n*: $d\ dvd\ n$
proof −
define q **where** $q = n\ div\ d$
define r **where** $r = n\ mod\ d$
have *q-nonneg*: $0 \leq q$

```
    using n-pos d-bounds by (simp add: q-def pos-imp-zdiv-nonneg-iff)
  have r-bounds: 0 ≤ r r < d
    using d-bounds by (simp-all add: r-def)
  have n-eq: n = q * d + r
    by (simp add: q-def r-def)
  show ?thesis
  proof (cases r = 0)
    case True
    then have n = q * d
      using n-eq by simp
    then show ?thesis
      unfolding dvd-def by (metis mult.commute)
  next
    case False
    then have r-pos: 0 < r
      using r-bounds by linarith
    have qd-bounds: 0 < q * d q * d < n
      using n-eq r-bounds r-pos d-bounds n-pos by linarith+
    have qd-mod-in: (int (nat q) * d) mod n ∈ H
      by (rule residue-add-closed-mult[OF n-pos zero add-closed d-in])
    have qd-in: q * d ∈ H
      using qd-mod-in q-nonneg qd-bounds n-pos by simp
    have r-mod-in: (0 - q * d) mod n ∈ H
      by (rule residue-add-closed-diff[OF n-pos zero add-closed zero qd-in])
    have neg-eq: (0 - q * d) mod n = r
    proof -
      have q * d = n - r
        using n-eq by simp
      then have (0 - q * d) mod n = (r - n) mod n
        by simp
      also have ... = r mod n
        by (simp add: mod-simps)
      also have ... = r
        using r-bounds d-bounds n-pos by simp
      finally show ?thesis .
    qed
    have r-in: r ∈ H
      using r-mod-in neg-eq by simp
    have r-inS: r ∈ ?S
      using r-in r-pos by auto
    have d ≤ r
      using Min-le[OF finS r-inS] by (simp add: d-def)
    with r-bounds show ?thesis
      by linarith
  qed
qed
have multiples-sub: {x ∈ {0..n - 1}. d dvd x} ⊆ H
proof
  fix x
  assume x-in: x ∈ {x ∈ {0..n - 1}. d dvd x}
  then have x-bounds: 0 ≤ x x < n and d-x: d dvd x
    by auto
  obtain q where x-eq: x = d * q
    using d-x unfolding dvd-def by blast
  have q-nonneg: 0 ≤ q
  proof (rule ccontr)
    assume ¬ 0 ≤ q
    then have q-neg: q < 0
      by simp
```

```
    have d * q < 0
      using d-bounds q-neg by (simp add: mult-less-0-iff)
    then have x < 0
      using x-eq by simp
    with x-bounds show False
      by linarith
  qed
  have (int (nat q) * d) mod n ∈ H
    by (rule residue-add-closed-mult[OF n-pos zero add-closed d-in])
  moreover have int (nat q) * d = x
    using q-nonneg x-eq by (simp add: mult.commute)
  ultimately show x ∈ H
    using x-bounds n-pos by simp
 qed
 show 0 < d
  by (rule d-bounds(1))
 show d dvd n
  by (rule d-dvd-n)
 show H = {x ∈ {0..n − 1}. d dvd x}
 proof
  show H ⊆ {x ∈ {0..n − 1}. d dvd x}
    using H-sub dvd-H by auto
  show {x ∈ {0..n − 1}. d dvd x} ⊆ H
    by (rule multiples-sub)
 qed
qed

lemma zmod-stabilizer-add-closed:
 assumes x-in: x ∈ Zmod.stabilizer p C
 assumes y-in: y ∈ Zmod.stabilizer p C
 shows (x + y) mod int p ∈ Zmod.stabilizer p C
proof −
 interpret H: subgroup Zmod.stabilizer p C {0..int (p − 1)}
    (λx y. (x + y) mod int p) 0::int
  by (rule Zmod.stabilizer-is-subgroup)
 show ?thesis
  by (rule H.sub-composition-closed[OF x-in y-in])
qed

lemma zmod-stabilizer-obtain-proper-divisor:
 fixes C :: int set
 assumes p-pos: 0 < p
 assumes nontrivial: Zmod.stabilizer p C ≠ {0}
 assumes proper: Zmod.stabilizer p C ≠ {0..int p − 1}
 obtains d where 1 < d d dvd int p
  ⋀h. h ∈ Zmod.stabilizer p C ⟹ d dvd h
proof −
 let ?H = Zmod.stabilizer p C
 have H-sub: ?H ⊆ {0..int p − 1}
  using Zmod.stabilizer-subset-group[of p C] p-pos by auto
 have zero: 0 ∈ ?H
  by (rule Zmod.zero-mem-stabilizer)
 have add-closed: ⋀x y. x ∈ ?H ⟹ y ∈ ?H ⟹ (x + y) mod int p ∈ ?H
  by (rule zmod-stabilizer-add-closed)
 show ?thesis
  by (rule residue-add-closed-obtain-proper-divisor
     [OF - H-sub zero add-closed nontrivial proper that])
    (use p-pos in simp)
qed
```

lemma *zmod-full-stabilizer-zero-imp-carrier-subset*:
fixes $C :: int\ set$
assumes *p-pos*: $0 < p$
assumes *C-sub*: $C \subseteq \{0..int\ p - 1\}$
assumes *zero-C*: $0 \in C$
assumes *full*: $Zmod.stabilizer\ p\ C = \{0..int\ p - 1\}$
shows $\{0..int\ p - 1\} \subseteq C$
proof
fix x
assume *x-carrier*: $x \in \{0..int\ p - 1\}$
then have *x-stab*: $x \in Zmod.stabilizer\ p\ C$
using *full* **by** *simp*
have *carrier-eq*: $\{0..int\ (p - 1)\} = \{0..int\ p - 1\}$
using *p-pos* **by** *auto*
have *zero-carrier*: $0 \in \{0..int\ (p - 1)\}$
using *p-pos* **by** *auto*
have *x-carrier'*: $x \in \{0..int\ (p - 1)\}$
using *x-carrier carrier-eq* **by** *simp*
have *C-sub'*: $C \subseteq \{0..int\ (p - 1)\}$
using *C-sub carrier-eq* **by** *simp*
have *coset-sub*: $Zmod.sumset\ p\ \{0\}\ (Zmod.stabilizer\ p\ C) \subseteq C$
by (*rule Zmod.stabilizer-coset-subset*[*OF C-sub' zero-C*])
have *x-sum*: $x \in Zmod.sumset\ p\ \{0\}\ (Zmod.stabilizer\ p\ C)$
proof −
have $(0 + x)\ mod\ int\ p \in Zmod.sumset\ p\ \{0\}\ (Zmod.stabilizer\ p\ C)$
by (*rule Zmod.sumset.sumsetI*) (*use x-stab x-carrier' zero-carrier* **in** *auto*)
then show *?thesis*
using *x-carrier p-pos* **by** *simp*
qed
show $x \in C$
using *coset-sub x-sum* **by** *auto*
qed

lemma *mod-image-int-normalized-interval*:
fixes $A :: int\ set$
assumes *n-pos*: $0 < n$
assumes *subset*: $A \subseteq \{0..n\}$
assumes *zero*: $0 \in A$
assumes *top*: $n \in A$
shows $mod\text{-}image\text{-}int\ n\ A = A - \{n\}$
proof
show $mod\text{-}image\text{-}int\ n\ A \subseteq A - \{n\}$
proof
fix r
assume $r \in mod\text{-}image\text{-}int\ n\ A$
then obtain a **where** *a-in*: $a \in A$ **and** *r-eq*: $r = a\ mod\ n$
by (*auto simp*: *mod-image-int-def*)
have *a-bounds*: $0 \leq a\ a \leq n$
using *subset a-in* **by** *auto*
show $r \in A - \{n\}$
proof (*cases* $a = n$)
case *True*
then show *?thesis*
using *zero r-eq n-pos* **by** *simp*
next
case *False*
then have $a < n$
using *a-bounds* **by** *simp*

```
      then have a mod n = a
        using a-bounds n-pos by simp
      then show ?thesis
        using a-in r-eq False by simp
    qed
  qed
next
  show A − {n} ⊆ mod-image-int n A
  proof
    fix a
    assume a-in: a ∈ A − {n}
    then have a ∈ A and a-ne: a ≠ n
      by auto
    have a-bounds: 0 ≤ a a ≤ n
      using subset ‹a ∈ A› by auto
    then have a < n
      using a-ne by simp
    then have a-mod: a mod n = a
      using a-bounds n-pos by simp
    show a ∈ mod-image-int n A
      unfolding mod-image-int-def
    proof (rule image-eqI)
      show a = a mod n
        using a-mod by simp
      show a ∈ A
        by fact
    qed
  qed
qed

lemma card-mod-image-int-normalized-interval:
  fixes A :: int set
  assumes fin: finite A
  assumes n-pos: 0 < n
  assumes subset: A ⊆ {0..n}
  assumes zero: 0 ∈ A
  assumes top: n ∈ A
  shows card (mod-image-int n A) = card A − 1
proof −
  have card (mod-image-int n A) = card (A − {n})
    by (simp add: mod-image-int-normalized-interval[OF n-pos subset zero top])
  also have ... = card A − 1
    using fin top by simp
  finally show ?thesis .
qed

lemma normalized-mod-sumset-trivial-stabilizer-card-ge:
  fixes A :: int set
  assumes fin: finite A
  assumes n-pos: 0 < n
  assumes subset: A ⊆ {0..n}
  assumes zero: 0 ∈ A
  assumes top: n ∈ A
  defines B ≡ mod-image-int n A
  defines p ≡ nat n
  defines C ≡ Zmod.sumset p B B
  assumes trivial: Zmod.stabilizer p C = {0}
  shows 2 ∗ card A − 3 ≤ card (mod-sumset-int n A A)
proof −
```

```
  have p-pos: 0 < p
    using n-pos by (simp add: p-def)
  have int-p: int p = n
    using n-pos by (simp add: p-def)
  have B-sub: B ⊆ {0..int p − 1}
    using mod-image-int-subset-residues[OF n-pos, of A] by (simp add: B-def int-p)
  have finB: finite B
    unfolding B-def by (rule finite-mod-image-int[OF fin])
  have zero-B: 0 ∈ B
    unfolding B-def mod-image-int-def
  proof (rule image-eqI)
    show 0 = 0 mod n
      by simp
    show 0 ∈ A
      by (rule zero)
  qed
  have nonemptyB: B ≠ {}
    using zero-B by auto
  have cardB: card B = card A − 1
    unfolding B-def by (rule card-mod-image-int-normalized-interval[OF fin n-pos subset zero top])
  have trivial': Zmod.stabilizer p (Zmod.sumset p B B) = {0}
    using trivial by (simp add: C-def)
  have C-lower-unfolded: 2 ∗ card B − 1 ≤ card (Zmod.sumset p B B)
    by (rule zmod-sumset-trivial-stabilizer-card-ge
        [OF p-pos B-sub finB nonemptyB trivial'])
  have C-lower: 2 ∗ card B − 1 ≤ card C
    using C-lower-unfolded by (simp add: C-def)
  have C-eq-mod: C = mod-sumset-int n A A
  proof −
    have C = mod-sumset-int (int p) B B
      unfolding C-def by (rule zmod-sumset-eq-mod-sumset-int[OF p-pos B-sub B-sub])
    also have ... = mod-sumset-int n B B
      by (simp add: int-p)
    also have ... = mod-sumset-int n A A
      unfolding B-def by (rule mod-sumset-int-mod-image-self[OF n-pos])
    finally show ?thesis .
  qed
  have 2 ∗ card A − 3 ≤ 2 ∗ card B − 1
    using cardB zero fin by (simp add: card-gt-0-iff)
  also have ... ≤ card C
    by (rule C-lower)
  finally show ?thesis
    by (simp add: C-eq-mod)
qed

lemma mod-image-int-subset-mod-sumset-self:
  assumes zero: 0 ∈ A
  shows mod-image-int n A ⊆ mod-sumset-int n A A
proof
  fix r
  assume r ∈ mod-image-int n A
  then obtain a where a-in: a ∈ A and r-eq: r = a mod n
    by (auto simp: mod-image-int-def)
  have 0 + a ∈ sumset A A
    using zero a-in by (rule sumsetI)
  then show r ∈ mod-sumset-int n A A
    using r-eq by (auto simp: mod-sumset-int-def mod-image-int-def)
qed
```

lemma *card-eq-sum-mod-fibers*:
fixes S :: *int set*
assumes *fin*: *finite S*
shows *card S* = $(\sum r{\in}$*mod-image-int n S*. *card* (*mod-fiber-int n S r*))
proof −
let *?C* = *mod-image-int n S*
have *finC*: *finite ?C*
by (*rule finite-mod-image-int*[*OF fin*])
have *union-eq*: *S* = $(\bigcup$ (*mod-fiber-int n S* ‘ *?C*))
by (*auto simp*: *mod-image-int-def mod-fiber-int-def*)
have *fin-fibers*: $\forall r{\in}$*?C*. *finite* (*mod-fiber-int n S r*)
using *fin* **by** *auto*
have *disj-fibers*:
$\forall r{\in}$*?C*. $\forall t{\in}$*?C*. *r* ≠ *t* ⟶
mod-fiber-int n S r ∩ *mod-fiber-int n S t* = {}
by (*auto simp*: *mod-fiber-int-def*)
have *card S* = *card* $(\bigcup$ (*mod-fiber-int n S* ‘ *?C*))
using *union-eq* **by** *simp*
also have ... = $(\sum r{\in}$*?C*. *card* (*mod-fiber-int n S r*))
by (*rule card-UN-disjoint*[*OF finC fin-fibers disj-fibers*])
finally show *?thesis* **.**
qed

lemma *card-mod-fiber-pos*:
fixes S :: *int set*
assumes *fin*: *finite S*
assumes *r-in*: *r* ∈ *mod-image-int n S*
shows *0* < *card* (*mod-fiber-int n S r*)
proof −
from *r-in* **obtain** *s* **where** *s-in*: *s* ∈ *S* **and** *r-eq*: *r* = *s mod n*
by (*auto simp*: *mod-image-int-def*)
have {*s*} ⊆ *mod-fiber-int n S r*
using *s-in r-eq* **by** (*auto simp*: *mod-fiber-int-def*)
then have *card* {*s*} ≤ *card* (*mod-fiber-int n S r*)
by (*rule card-mono*[*OF finite-mod-fiber-int*[*OF fin*]])
then show *?thesis*
by *simp*
qed

lemma *normalized-endpoint-zero-fiber-card-ge3*:
fixes A :: *int set*
assumes *fin*: *finite A*
assumes *n-pos*: *0* < *n*
assumes *zero*: *0* ∈ *A*
assumes *top*: *n* ∈ *A*
shows *3* ≤ *card* (*mod-fiber-int n* (*sumset A A*) *0*)
proof −
let *?S* = *sumset A A*
have *elems*: {*0*, *n*, *2* ∗ *n*} ⊆ *mod-fiber-int n ?S 0*
proof
fix *y*
assume *y* ∈ {*0*, *n*, *2* ∗ *n*}
then consider *y* = *0* | *y* = *n* | *y* = *2* ∗ *n*
by *auto*
then show *y* ∈ *mod-fiber-int n ?S 0*
proof *cases*
case *1*
have *0* + *0* ∈ *?S*
using *zero zero* **by** (*rule sumsetI*)

```
    then show ?thesis
      using 1 by (simp add: mod-fiber-int-def)
  next
    case 2
    have 0 + n ∈ ?S
      using zero top by (rule sumsetI)
    then show ?thesis
      using 2 n-pos by (simp add: mod-fiber-int-def)
  next
    case 3
    have n + n ∈ ?S
      using top top by (rule sumsetI)
    then show ?thesis
      using 3 n-pos by (simp add: mod-fiber-int-def)
  qed
qed
have card {0, n, 2 * n} ≤ card (mod-fiber-int n ?S 0)
  by (rule card-mono[OF finite-mod-fiber-int[OF finite-sumset[OF fin fin]] elems])
moreover have card {0, n, 2 * n} = 3
  using n-pos by auto
ultimately show ?thesis
  by simp
qed

lemma normalized-endpoint-nonzero-fiber-card-ge2:
fixes A :: int set
assumes fin: finite A
assumes n-pos: 0 < n
assumes subset: A ⊆ {0..n}
assumes zero: 0 ∈ A
assumes top: n ∈ A
assumes r-in: r ∈ mod-image-int n A
assumes r-ne: r ≠ 0
shows 2 ≤ card (mod-fiber-int n (sumset A A) r)
proof −
let ?S = sumset A A
from r-in obtain a where a-in: a ∈ A and r-eq: r = a mod n
  by (auto simp: mod-image-int-def)
have a-bounds: 0 ≤ a a ≤ n
  using subset a-in by auto
have a-ne-n: a ≠ n
  using r-eq r-ne n-pos by auto
then have a-lt: a < n
  using a-bounds by simp
have a-ne-0: a ≠ 0
  using r-eq r-ne by auto
have a-mod: a mod n = a
  using a-bounds a-lt n-pos by simp
have elems: {a, n + a} ⊆ mod-fiber-int n ?S r
proof
  fix y
  assume y ∈ {a, n + a}
  then show y ∈ mod-fiber-int n ?S r
  proof
    assume y-eq: y = a
    have 0 + a ∈ ?S
      using zero a-in by (rule sumsetI)
    then show ?thesis
      using y-eq r-eq a-mod by (simp add: mod-fiber-int-def)
```

```
    next
      assume y ∈ {n + a}
      then have y-eq: y = n + a
        by simp
      have n + a ∈ ?S
        using top a-in by (rule sumsetI)
      moreover have (n + a) mod n = r
        using r-eq a-mod n-pos by simp
      ultimately show ?thesis
        using y-eq by (simp add: mod-fiber-int-def)
    qed
  qed
  have card {a, n + a} ≤ card (mod-fiber-int n ?S r)
    by (rule card-mono[OF finite-mod-fiber-int[OF finite-sumset[OF fin fin]] elems])
  moreover have card {a, n + a} = 2
    using n-pos by auto
  ultimately show ?thesis
    by simp
qed

lemma card-sumset-ge-mod-sumset-plus-card:
  fixes A :: int set
  assumes fin: finite A
  assumes n-pos: 0 < n
  assumes subset: A ⊆ {0..n}
  assumes zero: 0 ∈ A
  assumes top: n ∈ A
  shows card (sumset A A) ≥ card (mod-sumset-int n A A) + card A
proof −
  let ?S = sumset A A
  let ?C = mod-sumset-int n A A
  let ?B = mod-image-int n A
  have finS: finite ?S
    by (rule finite-sumset[OF fin fin])
  have finC: finite ?C
    by (rule finite-mod-sumset-int[OF fin fin])
  have finB: finite ?B
    by (rule finite-mod-image-int[OF fin])
  have C-eq: ?C = mod-image-int n ?S
    by (simp add: mod-sumset-int-def)
  have B-sub-C: ?B ⊆ ?C
    by (rule mod-image-int-subset-mod-sumset-self[OF zero])
  have zero-in-B: 0 ∈ ?B
    unfolding mod-image-int-def
  proof (rule image-eqI)
    show 0 = 0 mod n
      by simp
    show 0 ∈ A
      by (rule zero)
  qed
  have cardB: card ?B = card A − 1
    by (rule card-mod-image-int-normalized-interval[OF fin n-pos subset zero top])
  have cardA-pos: 0 < card A
    using fin zero by (simp add: card-gt-0-iff, blast)
  have fiber-lower:
    1 + (if r ∈ ?B then 1 else 0) + (if r = 0 then 1 else 0)
      ≤ card (mod-fiber-int n ?S r) if r-in: r ∈ ?C for r
  proof (cases r = 0)
    case True
```

```
  have 3 ≤ card (mod-fiber-int n ?S r)
    using True by (simp add: normalized-endpoint-zero-fiber-card-ge3[OF fin n-pos zero top])
  then show ?thesis
    using True zero-in-B by simp
next
  case r-ne-zero: False
  show ?thesis
  proof (cases r ∈ ?B)
    case True
    have 2 ≤ card (mod-fiber-int n ?S r)
      by (rule normalized-endpoint-nonzero-fiber-card-ge2
          [OF fin n-pos subset zero top True r-ne-zero])
    then show ?thesis
      using True r-ne-zero by simp
  next
    case False
    have 0 < card (mod-fiber-int n ?S r)
      using r-in unfolding C-eq by (rule card-mod-fiber-pos[OF finS])
    then show ?thesis
      using False r-ne-zero by simp
  qed
qed
have extra-B-sum:
  (∑ r∈?C. (if r ∈ ?B then 1 else 0::nat)) = card ?B
proof −
  have (∑ r∈?C. (if r ∈ ?B then 1 else 0::nat)) =
      (∑ r∈?B. 1)
    using B-sub-C finB finC
    by (intro sum.mono-neutral-cong-right) auto
  then show ?thesis
    by simp
qed
have extra-zero-sum:
  (∑ r∈?C. (if r = 0 then 1 else 0::nat)) = 1
proof −
  have 0 ∈ ?C
    using zero-in-B B-sub-C by auto
  then show ?thesis
    using finC by (simp add: sum.If-cases)
qed
have extra-total:
  (∑ r∈?C. 1 + (if r ∈ ?B then 1 else 0) + (if r = 0 then 1 else 0)) =
    card ?C + card ?B + 1
proof −
  have sum-suc:
    (∑ r∈X. 1 + (if r ∈ ?B then 1 else 0) + (if r = 0 then 1 else 0)) =
      card X + (∑ r∈X. (if r ∈ ?B then 1 else 0) + (if r = 0 then 1 else 0))
    if finite X for X
    using that
  proof (induction X rule: finite-induct)
    case empty
    then show ?case
      by simp
  next
    case (insert r R)
    then show ?case
      by simp
  qed
  have split-extra:
```

```
    (∑ r∈?C. (if r ∈ ?B then 1 else 0) + (if r = 0 then 1 else 0)) =
      (∑ r∈?C. (if r ∈ ?B then 1 else 0)) +
      (∑ r∈?C. (if r = 0 then 1 else 0::nat))
    by (rule sum.distrib)
  show ?thesis
    using sum-suc[OF finC] split-extra extra-B-sum extra-zero-sum by simp
 qed
 have cardS-sum: card ?S = (∑ r∈?C. card (mod-fiber-int n ?S r))
  unfolding C-eq by (rule card-eq-sum-mod-fibers[OF finS])
 have sum-lower: (∑ r∈?C. card (mod-fiber-int n ?S r)) ≥
    (∑ r∈?C. 1 + (if r ∈ ?B then 1 else 0) + (if r = 0 then 1 else 0))
  by (rule sum-mono) (rule fiber-lower)
 have target-eq: card ?C + card ?B + 1 = card ?C + card A
  using cardB cardA-pos by linarith
 show ?thesis
  using cardS-sum sum-lower extra-total target-eq by linarith
qed

lemma normalized-small-doubling-mod-stabilizer-nontrivial:
 fixes A :: int set
 assumes fin: finite A
 assumes card-ge: 3 ≤ card A
 assumes n-pos: 0 < n
 assumes subset: A ⊆ {0..n}
 assumes zero: 0 ∈ A
 assumes top: n ∈ A
 assumes small-doubling: card (sumset A A) ≤ 3 * card A - 4
 defines B ≡ mod-image-int n A
 defines p ≡ nat n
 defines C ≡ Zmod.sumset p B B
 shows Zmod.stabilizer p C ≠ {0}
proof
 assume trivial: Zmod.stabilizer p C = {0}
 have trivial-unfolded:
  Zmod.stabilizer (nat n)
   (Zmod.sumset (nat n) (mod-image-int n A) (mod-image-int n A)) = {0}
  using trivial by (simp add: B-def p-def C-def)
 have mod-lower: 2 * card A - 3 ≤ card (mod-sumset-int n A A)
  by (rule normalized-mod-sumset-trivial-stabilizer-card-ge
     [OF fin n-pos subset zero top trivial-unfolded])
 have sum-lower:
  card (mod-sumset-int n A A) + card A ≤ card (sumset A A)
  by (rule card-sumset-ge-mod-sumset-plus-card[OF fin n-pos subset zero top])
 have 3 * card A - 3 ≤ card (sumset A A)
  using mod-lower sum-lower card-ge by linarith
 with small-doubling card-ge show False
  by linarith
qed

lemma normalized-small-doubling-obtain-positive-mod-period:
 fixes A :: int set
 assumes fin: finite A
 assumes card-ge: 3 ≤ card A
 assumes n-pos: 0 < n
 assumes subset: A ⊆ {0..n}
 assumes zero: 0 ∈ A
 assumes top: n ∈ A
 assumes small-doubling: card (sumset A A) ≤ 3 * card A - 4
 defines B ≡ mod-image-int n A
```

```
  defines p ≡ nat n
  defines C ≡ Zmod.sumset p B B
  obtains h where h ∈ Zmod.stabilizer p C 0 < h h < n
proof −
  have p-pos: 0 < p
    using n-pos by (simp add: p-def)
  have int-p: int p = n
    using n-pos by (simp add: p-def)
  have nontrivial-unfolded:
    Zmod.stabilizer (nat n)
      (Zmod.sumset (nat n) (mod-image-int n A) (mod-image-int n A)) ≠ {0}
    by (rule normalized-small-doubling-mod-stabilizer-nontrivial
        [OF fin card-ge n-pos subset zero top small-doubling])
  have nontrivial: Zmod.stabilizer p C ≠ {0}
    using nontrivial-unfolded by (simp add: B-def p-def C-def)
  obtain h where h-in: h ∈ Zmod.stabilizer p C and h-pos: 0 < h and h-lt-p: h < int p
    by (rule zmod-nontrivial-stabilizer-obtain-positive-period[OF p-pos nontrivial])
  have h-lt-n: h < n
    using h-lt-p by (simp add: int-p)
  show ?thesis
    by (rule that[OF h-in h-pos h-lt-n])
qed

lemma two-sided-unstable-hole-notin-mod-sumset:
  fixes A :: int set
  assumes n-pos: 0 < n
  assumes subset: A ⊆ {0..n}
  assumes x-bounds: 0 < x x < n
  assumes lower-missing: x ∉ sumset A A
  assumes upper-missing: n + x ∉ sumset A A
  shows x ∉ mod-sumset-int n A A
proof
  assume x ∈ mod-sumset-int n A A
  then obtain a b where a-in: a ∈ A and b-in: b ∈ A and mod-eq: x = (a + b) mod n
    by (auto simp: mod-sumset-int-iff)
  have a-bounds: 0 ≤ a a ≤ n
    using subset a-in by auto
  have b-bounds: 0 ≤ b b ≤ n
    using subset b-in by auto
  have sum-bounds: 0 ≤ a + b a + b ≤ 2 * n
    using a-bounds b-bounds by linarith+
  from mod-eq have (a + b) mod n = x
    by simp
  moreover have x mod n = x
    using n-pos x-bounds by simp
  ultimately have (a + b) mod n = x mod n
    by simp
  then obtain d where d-eq: x = (a + b) + n * d
    by (rule mod-eqE)
  define q where q = − d
  have q-eq: a + b = q * n + x
    using d-eq by (simp add: q-def algebra-simps)
  have q = 0 ∨ q = 1
  proof −
    have lower: 0 ≤ q * n + x
      using q-eq sum-bounds by simp
    have upper: q * n + x ≤ 2 * n
      using q-eq sum-bounds by simp
    have q-nonneg: 0 ≤ q
```

```
    proof (rule ccontr)
      assume ¬ 0 ≤ q
      then have q-le: q ≤ −1
        by linarith
      have q * n ≤ (−1) * n
        using q-le n-pos by (intro mult-right-mono) simp-all
      then have q * n + x < 0
        using x-bounds by linarith
      with lower show False
        by linarith
    qed
    have q-lt-two: q < 2
    proof (rule ccontr)
      assume ¬ q < 2
      then have q-ge: 2 ≤ q
        by linarith
      have 2 * n ≤ q * n
        using q-ge n-pos by (intro mult-right-mono) simp-all
      then have 2 * n < q * n + x
        using x-bounds by linarith
      with upper show False
        by linarith
    qed
    show ?thesis
      using q-nonneg q-lt-two
      by linarith
  qed
  then show False
  proof
    assume q = 0
    then have a + b = x
      using q-eq by simp
    moreover have a + b ∈ sumset A A
      using a-in b-in by (rule sumsetI)
    ultimately have x ∈ sumset A A
      by simp
    with lower-missing show False
      by simp
  next
    assume q = 1
    then have a + b = n + x
      using q-eq by simp
    moreover have a + b ∈ sumset A A
      using a-in b-in by (rule sumsetI)
    ultimately have n + x ∈ sumset A A
      by simp
    with upper-missing show False
      by simp
  qed
qed

lemma mod-sumset-gap-imp-stable-sum-hole:
  fixes A :: int set
  assumes fin: finite A
  assumes n-pos: 0 < n
  assumes subset: A ⊆ {0..n}
  assumes zero: 0 ∈ A
  assumes top: n ∈ A
  assumes x-bounds: 0 < x x < n
```

```
  assumes gap: x ∉ mod-sumset-int n A A
  shows x ∈ stable-sum-holes A
proof −
  have nonempty: A ≠ {}
    using top by auto
  have max-eq: Max A = n
    using Max-eq-iff[OF fin nonempty, of n] subset top by auto
  have x-mod: x mod n = x
    using n-pos x-bounds by simp
  have x-notin-A: x ∉ A
  proof
    assume x-in: x ∈ A
    have sum-x: 0 + x ∈ sumset A A
      using zero x-in by (rule sumsetI)
    have x ∈ mod-sumset-int n A A
      unfolding mod-sumset-int-def mod-image-int-def
    proof (rule image-eqI)
      show x = x mod n
        using x-mod by simp
      show x ∈ sumset A A
        using sum-x by simp
    qed
    with gap show False
      by simp
  qed
  have x-not-lower: x ∉ lower-sum-holes A
  proof
    assume x-lower: x ∈ lower-sum-holes A
    then have sum-x: x ∈ sumset A A
      by (simp add: lower-sum-holes-def)
    have x ∈ mod-sumset-int n A A
      unfolding mod-sumset-int-def mod-image-int-def
    proof (rule image-eqI)
      show x = x mod n
        using x-mod by simp
      show x ∈ sumset A A
        by (rule sum-x)
    qed
    with gap show False
      by simp
  qed
  have x-not-upper: x ∉ upper-sum-holes A
  proof
    assume x-upper: x ∈ upper-sum-holes A
    then have sum-nx: n + x ∈ sumset A A
      by (simp add: upper-sum-holes-def max-eq)
    have nx-mod: (n + x) mod n = x
      using n-pos x-mod by simp
    have x ∈ mod-sumset-int n A A
      unfolding mod-sumset-int-def mod-image-int-def
    proof (rule image-eqI)
      show x = (n + x) mod n
        using nx-mod by simp
      show n + x ∈ sumset A A
        by (rule sum-nx)
    qed
    with gap show False
      by simp
  qed
```

```
  have x ∈ interval-holes A
    using x-bounds x-notin-A by (auto simp: interval-holes-def max-eq)
  with x-not-lower x-not-upper show ?thesis
    by (auto simp: stable-sum-holes-def)
qed

lemma stable-sum-holes-eq-mod-sumset-gaps:
  fixes A :: int set
  assumes fin: finite A
  assumes n-pos: 0 < n
  assumes subset: A ⊆ {0..n}
  assumes zero: 0 ∈ A
  assumes top: n ∈ A
  shows stable-sum-holes A =
    {x. 0 < x ∧ x < n ∧ x ∉ mod-sumset-int n A A}
proof
  have nonempty: A ≠ {}
    using top by auto
  have max-eq: Max A = n
    using Max-eq-iff[OF fin nonempty, of n] subset top by auto
  show stable-sum-holes A ⊆
      {x. 0 < x ∧ x < n ∧ x ∉ mod-sumset-int n A A}
  proof
    fix x
    assume x-stable: x ∈ stable-sum-holes A
    then have x-hole: x ∈ interval-holes A
      by (auto simp: stable-sum-holes-def)
    then have x-nonneg: 0 ≤ x and x-le: x ≤ n and x-notin: x ∉ A
      by (auto simp: interval-holes-def max-eq)
    have x-ne-zero: x ≠ 0
      using x-notin zero by auto
    have x-pos: 0 < x
      using x-nonneg x-ne-zero by linarith
    have x-ne-n: x ≠ n
      using x-notin top by auto
    have x-lt: x < n
      using x-le x-ne-n by linarith
    have x-not-lower: x ∉ lower-sum-holes A
      using x-stable by (auto simp: stable-sum-holes-def)
    have x-not-upper: x ∉ upper-sum-holes A
      using x-stable by (auto simp: stable-sum-holes-def)
    have lower-missing: x ∉ sumset A A
    proof
      assume x ∈ sumset A A
      with x-hole have x ∈ lower-sum-holes A
        by (simp add: lower-sum-holes-def)
      with x-not-lower show False
        by simp
    qed
    have upper-missing: n + x ∉ sumset A A
    proof
      assume n + x ∈ sumset A A
      with x-hole have x ∈ upper-sum-holes A
        by (simp add: upper-sum-holes-def max-eq)
      with x-not-upper show False
        by simp
    qed
    have x ∉ mod-sumset-int n A A
      by (rule two-sided-unstable-hole-notin-mod-sumset
```

```
      [OF n-pos subset x-pos x-lt lower-missing upper-missing])
    with x-pos x-lt show x ∈ {x. 0 < x ∧ x < n ∧ x ∉ mod-sumset-int n A A}
      by simp
  qed
  show {x. 0 < x ∧ x < n ∧ x ∉ mod-sumset-int n A A}
    ⊆ stable-sum-holes A
  proof
    fix x
    assume x ∈ {x. 0 < x ∧ x < n ∧ x ∉ mod-sumset-int n A A}
    then have x-bounds: 0 < x x < n and gap: x ∉ mod-sumset-int n A A
      by auto
    show x ∈ stable-sum-holes A
    by (rule mod-sumset-gap-imp-stable-sum-hole
      [OF fin n-pos subset zero top x-bounds gap])
  qed
qed

lemma mod-sumset-int-eq-mod-image-union-sum-holes:
  fixes A :: int set
  assumes fin: finite A
  assumes n-pos: 0 < n
  assumes subset: A ⊆ {0..n}
  assumes zero: 0 ∈ A
  assumes top: n ∈ A
  assumes max-eq: Max A = n
  shows mod-sumset-int n A A =
    mod-image-int n A ∪ lower-sum-holes A ∪ upper-sum-holes A
proof
  let ?C = mod-sumset-int n A A
  let ?B = mod-image-int n A
  have stable-eq:
    stable-sum-holes A =
      {x. 0 < x ∧ x < n ∧ x ∉ ?C}
    by (rule stable-sum-holes-eq-mod-sumset-gaps[OF fin n-pos subset zero top])
  show ?C ⊆ ?B ∪ lower-sum-holes A ∪ upper-sum-holes A
  proof
    fix r
    assume r-in: r ∈ ?C
    have r-res: r ∈ {0..n - 1}
      using r-in mod-image-int-subset-residues[OF n-pos, of sumset A A]
      by (auto simp: mod-sumset-int-def)
    show r ∈ ?B ∪ lower-sum-holes A ∪ upper-sum-holes A
    proof (cases r ∈ ?B)
      case True
      then show ?thesis
        by simp
    next
      case False
      have r-not-A: r ∉ A
      proof
        assume r-A: r ∈ A
        have r-mod: r mod n = r
          using r-res n-pos by simp
        have r ∈ ?B
          unfolding mod-image-int-def
        proof (rule image-eqI)
          show r = r mod n
            using r-mod by simp
          show r ∈ A
```

```
        by (rule r-A)
    qed
    with False show False
      by simp
  qed
  have r-hole: r ∈ interval-holes A
    using r-res r-not-A by (auto simp: interval-holes-def max-eq)
  have r ∈ lower-sum-holes A ∪ upper-sum-holes A
  proof (rule ccontr)
    assume r ∉ lower-sum-holes A ∪ upper-sum-holes A
    then have r ∈ stable-sum-holes A
      using r-hole by (auto simp: stable-sum-holes-def)
    then have r ∉ ?C
      using stable-eq by auto
    with r-in show False
      by simp
  qed
  then show ?thesis
    by simp
 qed
qed
show ?B ∪ lower-sum-holes A ∪ upper-sum-holes A ⊆ ?C
proof
 fix r
 assume r-in: r ∈ ?B ∪ lower-sum-holes A ∪ upper-sum-holes A
 then show r ∈ ?C
 proof
  assume r ∈ ?B ∪ lower-sum-holes A
  then show ?thesis
  proof
   assume r ∈ ?B
   then show ?thesis
    using mod-image-int-subset-mod-sumset-self[OF zero] by auto
  next
   assume r-lower: r ∈ lower-sum-holes A
   then have r-sum: r ∈ sumset A A
    by (simp add: lower-sum-holes-def)
   have r-bounds: 0 ≤ r r < n
   proof −
    have r-hole: r ∈ interval-holes A
      using r-lower by (simp add: lower-sum-holes-def)
    then have 0 ≤ r r ≤ n r ∉ A
      by (auto simp: interval-holes-def max-eq)
    moreover have r ≠ n
      using ‹r ∉ A› top by auto
    ultimately show 0 ≤ r r < n
      by linarith+
   qed
   have r-mod: r mod n = r
    using r-bounds n-pos by simp
   show ?thesis
    unfolding mod-sumset-int-def mod-image-int-def
   proof (rule image-eqI)
    show r = r mod n
      using r-mod by simp
    show r ∈ sumset A A
      by (rule r-sum)
   qed
  qed
```

```
    next
      assume r-upper: r ∈ upper-sum-holes A
      then have nr-sum: n + r ∈ sumset A A
        by (simp add: upper-sum-holes-def max-eq)
      have r-bounds: 0 ≤ r r < n
      proof −
        have r-hole: r ∈ interval-holes A
          using r-upper by (simp add: upper-sum-holes-def)
        then have 0 ≤ r r ≤ n r ∉ A
          by (auto simp: interval-holes-def max-eq)
        moreover have r ≠ n
          using ‹r ∉ A› top by auto
        ultimately show 0 ≤ r r < n
          by linarith+
      qed
      have nr-mod: (n + r) mod n = r
        using r-bounds n-pos by simp
      show ?thesis
        unfolding mod-sumset-int-def mod-image-int-def
      proof (rule image-eqI)
        show r = (n + r) mod n
          using nr-mod by simp
        show n + r ∈ sumset A A
          by (rule nr-sum)
      qed
    qed
  qed
qed

lemma card-mod-sumset-int-eq-card-mod-image-plus-unstable-holes:
  fixes A :: int set
  assumes fin: finite A
  assumes n-pos: 0 < n
  assumes subset: A ⊆ {0..n}
  assumes zero: 0 ∈ A
  assumes top: n ∈ A
  assumes max-eq: Max A = n
  shows card (mod-sumset-int n A A) =
    card (mod-image-int n A) + card (lower-sum-holes A ∪ upper-sum-holes A)
proof −
  let ?B = mod-image-int n A
  let ?U = lower-sum-holes A ∪ upper-sum-holes A
  have C-eq: mod-sumset-int n A A = ?B ∪ ?U
    using mod-sumset-int-eq-mod-image-union-sum-holes
      [OF fin n-pos subset zero top max-eq]
    by auto
  have finB: finite ?B
    by (rule finite-mod-image-int[OF fin])
  have finU: finite ?U
    by simp
  have disj: ?B ∩ ?U = {}
  proof
    show ?B ∩ ?U ⊆ {}
    proof
      fix r
      assume r-in: r ∈ ?B ∩ ?U
      then have r-hole: r ∈ interval-holes A
        by (auto simp: lower-sum-holes-def upper-sum-holes-def)
      then have r-not-A: r ∉ A
```

```
      by (simp add: interval-holes-def)
    from r-in obtain a where a-in: a ∈ A and r-eq: r = a mod n
      by (auto simp: mod-image-int-def)
    have a-bounds: 0 ≤ a a ≤ n
      using subset a-in by auto
    show r ∈ {}
    proof (cases a = n)
      case True
      then have r = 0
        using r-eq n-pos by simp
      with zero r-not-A show ?thesis
        by simp
    next
      case False
      then have a < n
        using a-bounds by simp
      then have a mod n = a
        using a-bounds n-pos by simp
      with r-eq a-in r-not-A show ?thesis
        by simp
    qed
  qed
  qed simp
  have card (mod-sumset-int n A A) = card (?B ∪ ?U)
    by (simp add: C-eq)
  also have ... = card ?B + card ?U
    by (rule card-Un-disjoint[OF finB finU disj])
  finally show ?thesis .
qed

lemma card-mod-sumset-int-eq-card-A-minus-one-plus-unstable-holes:
  fixes A :: int set
  assumes fin: finite A
  assumes n-pos: 0 < n
  assumes subset: A ⊆ {0..n}
  assumes zero: 0 ∈ A
  assumes top: n ∈ A
  assumes max-eq: Max A = n
  shows card (mod-sumset-int n A A) =
    card A − 1 + card (lower-sum-holes A ∪ upper-sum-holes A)
proof −
  have card-mod:
    card (mod-sumset-int n A A) =
      card (mod-image-int n A) + card (lower-sum-holes A ∪ upper-sum-holes A)
    by (rule card-mod-sumset-int-eq-card-mod-image-plus-unstable-holes
        [OF fin n-pos subset zero top max-eq])
  have card-B: card (mod-image-int n A) = card A − 1
    by (rule card-mod-image-int-normalized-interval[OF fin n-pos subset zero top])
  show ?thesis
    using card-mod card-B by simp
qed

lemma normalized-sumset-card-eq-mod-sumset-plus-card-inter-holes:
  fixes A :: int set
  assumes fin: finite A
  assumes n-pos: 0 < n
  assumes subset: A ⊆ {0..n}
  assumes zero: 0 ∈ A
  assumes top: n ∈ A
```

```
  assumes max-eq: Max A = n
  assumes nonneg: ⋀x. x ∈ A ⟹ 0 ≤ x
  shows card (sumset A A) =
    card (mod-sumset-int n A A) + card A +
    card (lower-sum-holes A ∩ upper-sum-holes A)
proof −
  let ?L = lower-sum-holes A
  let ?U = upper-sum-holes A
  have sum-card:
    card (sumset A A) = 2 ∗ card A − 1 + card ?L + card ?U
    by (rule normalized-sumset-card-eq-with-holes[OF fin zero nonneg])
  have mod-card:
    card (mod-sumset-int n A A) = card A − 1 + card (?L ∪ ?U)
    by (rule card-mod-sumset-int-eq-card-A-minus-one-plus-unstable-holes
        [OF fin n-pos subset zero top max-eq])
  have union-inter:
    card ?L + card ?U = card (?L ∪ ?U) + card (?L ∩ ?U)
    by (rule card-Un-Int) simp-all
  show ?thesis
    using sum-card mod-card union-inter by linarith
qed

lemma card-mod-sumset-int-eq-diameter-minus-stable-holes:
  fixes A :: int set
  assumes fin: finite A
  assumes n-pos: 0 < n
  assumes subset: A ⊆ {0..n}
  assumes zero: 0 ∈ A
  assumes top: n ∈ A
  assumes max-eq: Max A = n
  shows card (mod-sumset-int n A A) = nat n − card (stable-sum-holes A)
proof −
  let ?C = mod-sumset-int n A A
  let ?R = {0..n − 1}
  let ?G = {x. 0 < x ∧ x < n ∧ x ∉ ?C}
  have C-sub: ?C ⊆ ?R
    unfolding mod-sumset-int-def by (rule mod-image-int-subset-residues[OF n-pos])
  have zero-C: 0 ∈ ?C
    using zero unfolding mod-sumset-int-iff
    by (rule-tac x = 0 in bexI, rule-tac x = 0 in bexI) simp-all
  have R-minus-C: ?R − ?C = ?G
  proof
    show ?R − ?C ⊆ ?G
    proof
      fix x
      assume x-in: x ∈ ?R − ?C
      then have x-bounds: 0 ≤ x x < n
        by auto
      have x-ne-zero: x ≠ 0
        using x-in zero-C by auto
      with x-bounds x-in show x ∈ ?G
        by auto
    qed
    show ?G ⊆ ?R − ?C
    proof
      fix x
      assume x ∈ ?G
      then show x ∈ ?R − ?C
        by auto
```

```
    qed
  qed
  have stable-eq: stable-sum-holes A = ?G
    using stable-sum-holes-eq-mod-sumset-gaps[OF fin n-pos subset zero top] max-eq by simp
  have finC: finite ?C
    by (rule finite-mod-sumset-int[OF fin fin])
  have card-diff: card (?R − ?C) = card ?R − card ?C
    by (rule card-Diff-subset[OF finC C-sub])
  have cardC-le: card ?C ≤ card ?R
    by (rule card-mono[OF - C-sub]) simp
  have cardR: card ?R = nat n
    using n-pos by simp
  have card-stable: card (stable-sum-holes A) = nat n − card ?C
    using card-diff cardR stable-eq R-minus-C by simp
  show ?thesis
    using card-stable cardC-le cardR by simp
qed

lemma mod-sub-translate-inj-on-residues:
  fixes x n :: int
  assumes n-pos: 0 < n
  shows inj-on (λy. (x − y) mod n) {0..n − 1}
proof (rule inj-onI)
  fix a b
  assume a-in: a ∈ {0..n − 1}
  assume b-in: b ∈ {0..n − 1}
  assume eq: (x − a) mod n = (x − b) mod n
  have a-bounds: 0 ≤ a a < n
    using a-in by auto
  have b-bounds: 0 ≤ b b < n
    using b-in by auto
  have dvd-ba: n dvd b − a
    using eq by (simp add: mod-eq-dvd-iff algebra-simps)
  then obtain q where q-eq: b − a = n * q
    by (auto elim: dvdE)
  have diff-lower: − n < b − a
    using a-bounds b-bounds by linarith
  have diff-upper: b − a < n
    using a-bounds b-bounds by linarith
  have q = 0
  proof (rule ccontr)
    assume q ≠ 0
    then consider 1 ≤ q | q ≤ −1
      by linarith
    then show False
    proof cases
      case 1
      then have n ≤ n * q
      proof −
        have 1 * n ≤ q * n
          using 1 n-pos by (intro mult-right-mono) simp-all
        then show ?thesis
          by (simp add: mult.commute)
      qed
      then show False
        using q-eq diff-upper by linarith
    next
      case 2
      then have n * q ≤ − n
```

```
    proof −
      have q * n ≤ (−1) * n
        using 2 n-pos by (intro mult-right-mono) simp-all
      then show ?thesis
        by (simp add: mult.commute)
    qed
    then show False
      using q-eq diff-lower by linarith
  qed
 qed
 then show a = b
  using q-eq by simp
qed

lemma card-mod-sub-translate-eq:
 fixes A :: int set
 fixes x n :: int
 assumes n-pos: 0 < n
 assumes A-sub: A ⊆ {0..n − 1}
 shows card ((λy. (x − y) mod n) ‘ A) = card A
proof −
 have finA: finite A
  using A-sub by (rule finite-subset) simp
 have inj-res: inj-on (λy. (x − y) mod n) {0..n − 1}
  by (rule mod-sub-translate-inj-on-residues[OF n-pos])
 have inj-A: inj-on (λy. (x − y) mod n) A
  by (rule inj-on-subset[OF inj-res A-sub])
 show ?thesis
  by (simp add: card-image finA inj-A)
qed

lemma stable-mod-gap-translate-disjoint:
 fixes A :: int set
 assumes n-pos: 0 < n
 assumes x-mod: x mod n = x
 assumes gap: x ∉ mod-sumset-int n A A
 shows (λy. (x − y) mod n) ‘ mod-image-int n A ∩ mod-image-int n A = {}
proof
 show (λy. (x − y) mod n) ‘ mod-image-int n A ∩ mod-image-int n A ⊆ {}
 proof
  fix r
  assume r-in: r ∈ (λy. (x − y) mod n) ‘ mod-image-int n A ∩ mod-image-int n A
  then obtain a b where a-in: a ∈ A and b-in: b ∈ A
   and r-eq: r = (x − (a mod n)) mod n
   and r-mod: r = b mod n
   by (auto simp: mod-image-int-def)
  have x-sum-mod:
   x mod n = ((a mod n) + (b mod n)) mod n
   using r-eq r-mod by (simp add: mod-simps)
  have sum-mod: x mod n = (a + b) mod n
   using x-sum-mod by (simp add: mod-simps)
  have x mod n ∈ mod-sumset-int n A A
   using a-in b-in sum-mod by (auto simp: mod-sumset-int-iff)
  moreover have x mod n = x
   by (rule x-mod)
  ultimately have x ∈ mod-sumset-int n A A
   by simp
  with gap show r ∈ {}
   by simp
```

```
  qed
qed simp

lemma stable-mod-gap-card-le-half-diameter:
  fixes A :: int set
  assumes fin: finite A
  assumes n-pos: 0 < n
  assumes subset: A ⊆ {0..n}
  assumes zero: 0 ∈ A
  assumes top: n ∈ A
  assumes x-mod: x mod n = x
  assumes gap: x ∉ mod-sumset-int n A A
  shows 2 * (card A - 1) ≤ nat n
proof -
  let ?B = mod-image-int n A
  let ?T = (λy. (x - y) mod n) ` ?B
  have B-sub: ?B ⊆ {0..n - 1}
    by (rule mod-image-int-subset-residues[OF n-pos])
  have T-sub: ?T ⊆ {0..n - 1}
    using n-pos by (auto intro!: pos-mod-bound)
  have disj: ?T ∩ ?B = {}
    by (rule stable-mod-gap-translate-disjoint[OF n-pos x-mod gap])
  have cardT: card ?T = card ?B
    by (rule card-mod-sub-translate-eq[OF n-pos B-sub])
  have cardB: card ?B = card A - 1
    by (rule card-mod-image-int-normalized-interval[OF fin n-pos subset zero top])
  have union-sub: ?T ∪ ?B ⊆ {0..n - 1}
    using T-sub B-sub by auto
  have finT: finite ?T
    using T-sub by (rule finite-subset) simp
  have finB: finite ?B
    by (rule finite-mod-image-int[OF fin])
  have card (?T ∪ ?B) = card ?T + card ?B
    by (rule card-Un-disjoint[OF finT finB disj])
  also have ... = 2 * (card A - 1)
    by (simp add: cardT cardB)
  finally have card-union: card (?T ∪ ?B) = 2 * (card A - 1) .
  have card (?T ∪ ?B) ≤ card {0..n - 1}
    by (rule card-mono[OF - union-sub]) simp
  also have ... = nat n
    using n-pos by simp
  finally show ?thesis
    using card-union by simp
qed

lemma stable-sum-holes-empty-of-short-diameter:
  fixes A :: int set
  assumes fin: finite A
  assumes card-ge: 3 ≤ card A
  assumes zero: 0 ∈ A
  assumes nonneg: ⋀x. x ∈ A ⟹ 0 ≤ x
  assumes short: nat (Max A) ≤ 2 * card A - 3
  shows stable-sum-holes A = {}
proof (rule ccontr)
  assume stable-nonempty: stable-sum-holes A ≠ {}
  then obtain x where x-stable: x ∈ stable-sum-holes A
    by auto
  let ?n = Max A
  have nonempty: A ≠ {}
```

```
    using zero by auto
  have top: ?n ∈ A
    using fin nonempty by simp
  have subset: A ⊆ {0..?n}
    by (rule normalized-subset-interval[OF fin zero nonneg])
  have max-pos: 0 < ?n
  proof −
    have 1 < card A
      using card-ge by simp
    then have A ≠ {0}
      using zero fin by auto
    then obtain y where y-in: y ∈ A and y-ne: y ≠ 0
      using zero by auto
    have 0 < y
      using nonneg[OF y-in] y-ne by linarith
    also have y ≤ ?n
      using fin y-in by simp
    finally show ?thesis .
  qed
  have stable-eq:
    stable-sum-holes A =
      {x. 0 < x ∧ x < ?n ∧ x ∉ mod-sumset-int ?n A A}
    by (rule stable-sum-holes-eq-mod-sumset-gaps[OF fin max-pos subset zero top])
  have gap: x ∉ mod-sumset-int ?n A A
    using x-stable stable-eq by auto
  have x-mod: x mod ?n = x
    using x-stable stable-eq max-pos by auto
  have lower: 2 ∗ (card A − 1) ≤ nat ?n
    by (rule stable-mod-gap-card-le-half-diameter[OF fin max-pos subset zero top x-mod gap])
  have card-pos: 0 < card A
    using card-ge by simp
  have 2 ∗ card A − 2 ≤ nat ?n
    using lower card-pos by linarith
  with short card-ge show False
    by linarith
qed

lemma stable-sum-holes-nonempty-imp-large-diameter:
  fixes A :: int set
  assumes fin: finite A
  assumes card-ge: 3 ≤ card A
  assumes zero: 0 ∈ A
  assumes nonneg: ⋀x. x ∈ A ⟹ 0 ≤ x
  assumes stable-nonempty: stable-sum-holes A ≠ {}
  shows 2 ∗ card A − 2 ≤ nat (Max A)
proof (rule ccontr)
  assume ¬ 2 ∗ card A − 2 ≤ nat (Max A)
  then have short: nat (Max A) ≤ 2 ∗ card A − 3
    using card-ge by linarith
  have stable-sum-holes A = {}
    by (rule stable-sum-holes-empty-of-short-diameter[OF fin card-ge zero nonneg short])
  with stable-nonempty show False
    by simp
qed
```

4.6 Normalization by the diameter gcd

```
definition shifted-int-set :: int set ⇒ int set where
  shifted-int-set A = (λx. x − Min A) ‘ A
```

definition *int-set-content* :: *int set* ⇒ *int* **where**
int-set-content A = *Gcd* (*shifted-int-set A*)

definition *normalized-int-set* :: *int set* ⇒ *int set* **where**
normalized-int-set A = (λ*x*. *x div int-set-content A*) ‘ *shifted-int-set A*

lemma *finite-shifted-int-set* [*intro*]:
assumes *finite A*
shows *finite* (*shifted-int-set A*)
using *assms* **by** (*simp add*: *shifted-int-set-def*)

lemma *finite-normalized-int-set* [*intro*]:
assumes *finite A*
shows *finite* (*normalized-int-set A*)
using *assms* **by** (*simp add*: *normalized-int-set-def shifted-int-set-def*)

lemma *zero-in-shifted-int-set*:
assumes *finite A* **and** *A* ≠ {}
shows *0* ∈ *shifted-int-set A*
using *assms* **by** (*auto simp*: *shifted-int-set-def*)

lemma *int-set-content-dvd-shift*:
assumes *x* ∈ *A*
shows *int-set-content A dvd x* − *Min A*
using *assms* **by** (*auto simp*: *int-set-content-def shifted-int-set-def*)

lemma *int-set-content-dvd-shifted*:
assumes *s* ∈ *shifted-int-set A*
shows *int-set-content A dvd s*
using *assms* **by** (*simp add*: *int-set-content-def*)

lemma *card-ge2-imp-Min-less-Max*:
assumes *fin*: *finite A* **and** *card-ge2*: *2* ≤ *card A*
shows *Min A* < *Max A*
proof −
have *nonempty*: *A* ≠ {}
using *assms* **by** *auto*
have *min-in*: *Min A* ∈ *A*
using *fin nonempty* **by** *simp*
have ¬ *A* ⊆ {*Min A*}
proof
assume *A* ⊆ {*Min A*}
with *min-in* **have** *A-singleton*: *A* = {*Min A*}
by *auto*
have *card A* = *card* {*Min A*}
by (*rule arg-cong*[*OF A-singleton*])
also have ... = *1*
by *simp*
finally have *card A* = *1* **.**
with *card-ge2* **show** *False*
by *simp*
qed
then obtain *y* **where** *y-in*: *y* ∈ *A* **and** *y-ne*: *y* ≠ *Min A*
by *auto*
have *Min A* ≤ *y*
using *fin y-in* **by** *simp*
with *y-ne* **have** *Min A* < *y*
by *simp*

```
  also have y ≤ Max A
    using fin y-in by simp
  finally show ?thesis .
qed

lemma int-set-content-pos:
  assumes fin: finite A and card-ge2: 2 ≤ card A
  shows 0 < int-set-content A
proof −
  have nonempty: A ≠ {}
    using assms by auto
  have max-in: Max A ∈ A
    using fin nonempty by simp
  have min-lt-max: Min A < Max A
    by (rule card-ge2-imp-Min-less-Max[OF fin card-ge2])
  have max-shift: Max A − Min A ∈ shifted-int-set A
    using max-in by (auto simp: shifted-int-set-def)
  have Max A − Min A ≠ 0
    using min-lt-max by simp
  then have ¬ shifted-int-set A ⊆ {0}
    using max-shift by auto
  then have int-set-content A ≠ 0
    by (simp add: int-set-content-def)
  moreover have 0 ≤ int-set-content A
    by (simp add: int-set-content-def)
  ultimately show ?thesis
    by simp
qed

lemma normalized-int-set-reconstruction:
  assumes fin: finite A and card-ge2: 2 ≤ card A
  shows affine-image-int (Min A) (int-set-content A) (normalized-int-set A) = A
proof
  let ?g = int-set-content A
  show affine-image-int (Min A) ?g (normalized-int-set A) ⊆ A
  proof
    fix y
    assume y ∈ affine-image-int (Min A) ?g (normalized-int-set A)
    then obtain z where z-in: z ∈ normalized-int-set A and y-eq: y = Min A + ?g ∗ z
      by (auto simp: affine-image-int-def)
    from z-in obtain s where s-in: s ∈ shifted-int-set A and z-eq: z = s div ?g
      by (auto simp: normalized-int-set-def)
    from s-in obtain x where x-in: x ∈ A and s-eq: s = x − Min A
      by (auto simp: shifted-int-set-def)
    have dvd: ?g dvd x − Min A
      by (rule int-set-content-dvd-shift[OF x-in])
    have ?g ∗ ((x − Min A) div ?g) = x − Min A
      using dvd by simp
    then have y = x
      using y-eq z-eq s-eq by simp
    with x-in show y ∈ A
      by simp
  qed
next
  let ?g = int-set-content A
  show A ⊆ affine-image-int (Min A) ?g (normalized-int-set A)
  proof
    fix x
    assume x-in: x ∈ A
```

```
    have shift-in: x − Min A ∈ shifted-int-set A
      using x-in by (auto simp: shifted-int-set-def)
    have norm-in: (x − Min A) div ?g ∈ normalized-int-set A
      using shift-in by (auto simp: normalized-int-set-def)
    have dvd: ?g dvd x − Min A
      by (rule int-set-content-dvd-shift[OF x-in])
    have ?g * ((x − Min A) div ?g) = x − Min A
      using dvd by simp
    then have x = Min A + ?g * ((x − Min A) div ?g)
      by simp
    with norm-in show x ∈ affine-image-int (Min A) ?g (normalized-int-set A)
      by (auto simp: affine-image-int-def)
  qed
qed

lemma card-normalized-int-set:
  assumes fin: finite A and card-ge2: 2 ≤ card A
  shows card (normalized-int-set A) = card A
proof −
  let ?g = int-set-content A
  have finN: finite (normalized-int-set A)
    by (rule finite-normalized-int-set[OF fin])
  have g-nonzero: ?g ≠ 0
    using int-set-content-pos[OF fin card-ge2] by simp
  have card A = card (affine-image-int (Min A) ?g (normalized-int-set A))
    using normalized-int-set-reconstruction[OF fin card-ge2] by simp
  also have ... = card (normalized-int-set A)
    by (rule card-affine-image-int[OF finN g-nonzero])
  finally show ?thesis
    by simp
qed

lemma card-sumset-normalized-int-set:
  assumes fin: finite A and card-ge2: 2 ≤ card A
  shows card (sumset (normalized-int-set A) (normalized-int-set A)) =
    card (sumset A A)
proof −
  let ?N = normalized-int-set A
  let ?g = int-set-content A
  have finN: finite ?N
    by (rule finite-normalized-int-set[OF fin])
  have g-nonzero: ?g ≠ 0
    using int-set-content-pos[OF fin card-ge2] by simp
  have A-eq: A = affine-image-int (Min A) ?g ?N
    using normalized-int-set-reconstruction[OF fin card-ge2] by simp
  let ?A' = affine-image-int (Min A) ?g ?N
  have sum-eq1: sumset A A = sumset ?A' A
    using A-eq by (rule arg-cong[where f = λX. sumset X A])
  have sum-eq2: sumset ?A' A = sumset ?A' ?A'
    using A-eq by (rule arg-cong[where f = λX. sumset ?A' X])
  have card (sumset A A) = card (sumset ?A' ?A')
    using sum-eq1 sum-eq2 by simp
  also have ... = card (sumset ?N ?N)
    by (rule card-sumset-affine-image-int-self[OF finN g-nonzero])
  finally show ?thesis
    by simp
qed

lemma zero-in-normalized-int-set:
```

```
  assumes finite A and A ≠ {}
  shows 0 ∈ normalized-int-set A
proof −
  have 0 div int-set-content A ∈ normalized-int-set A
    unfolding normalized-int-set-def
  proof (rule image-eqI)
    show 0 div int-set-content A = 0 div int-set-content A
      by simp
    show 0 ∈ shifted-int-set A
      by (rule zero-in-shifted-int-set[OF assms])
  qed
  then show ?thesis
    by simp
qed

lemma normalized-int-set-nonneg:
  assumes fin: finite A and card-ge2: 2 ≤ card A
  assumes x-in: x ∈ normalized-int-set A
  shows 0 ≤ x
proof −
  let ?g = int-set-content A
  from x-in obtain s where s-in: s ∈ shifted-int-set A and x-eq: x = s div ?g
    by (auto simp: normalized-int-set-def)
  from s-in obtain a where a-in: a ∈ A and s-eq: s = a − Min A
    by (auto simp: shifted-int-set-def)
  have Min A ≤ a
    using fin a-in by simp
  then have 0 ≤ s
    using s-eq by simp
  moreover have 0 < ?g
    by (rule int-set-content-pos[OF fin card-ge2])
  ultimately show ?thesis
    using x-eq by (simp add: pos-imp-zdiv-nonneg-iff)
qed

lemma Gcd-normalized-int-set:
  assumes fin: finite A and card-ge2: 2 ≤ card A
  shows Gcd (normalized-int-set A) = 1
proof −
  let ?N = normalized-int-set A
  let ?g = int-set-content A
  have g-nonzero: ?g ≠ 0
    using int-set-content-pos[OF fin card-ge2] by simp
  have one-eq: (1 :: int) = Gcd ?N
  proof (rule GcdI)
    fix n
    assume n ∈ ?N
    show (1 :: int) dvd n
      by simp
  next
    fix c :: int
    assume c-dvd: ⋀n. n ∈ ?N ⟹ c dvd n
    have gc-dvd-shift: ?g * c dvd s if s-in: s ∈ shifted-int-set A for s
    proof −
      have div-in: s div ?g ∈ ?N
        using s-in by (auto simp: normalized-int-set-def)
      then have c-dvd-div: c dvd s div ?g
        by (rule c-dvd)
      obtain k where k: s div ?g = c * k
```

```
      using c-dvd-div unfolding dvd-def by blast
    have g-dvd-s: ?g dvd s
      by (rule int-set-content-dvd-shifted[OF s-in])
    have s = ?g * (s div ?g)
      using g-dvd-s by simp
    also have ... = (?g * c) * k
      using k by (simp add: algebra-simps)
    finally show ?thesis
      unfolding dvd-def by blast
  qed
  have ?g * c dvd Gcd (shifted-int-set A)
    by (rule Gcd-greatest) (rule gc-dvd-shift)
  then have ?g * c dvd ?g
    by (simp add: int-set-content-def)
  then have ?g * c dvd ?g * 1
    by simp
  with g-nonzero show c dvd (1 :: int)
    by simp
next
  show normalize (1 :: int) = 1
    by simp
qed
then show ?thesis
  by simp
qed

lemma group-closure-eq-UNIV-of-Gcd-one:
  fixes A :: int set
  assumes Gcd A = 1
  shows group-closure A = UNIV
  using assms by (auto simp: group-closure-eq)

lemma common-divisor-dvd-one-of-Gcd-one:
  fixes A :: int set
  assumes gcd-one: Gcd A = 1
  assumes dvd-all: ⋀a. a ∈ A ⟹ d dvd a
  shows d dvd (1 :: int)
proof −
  have d dvd Gcd A
    by (rule Gcd-greatest) (rule dvd-all)
  then show ?thesis
    using gcd-one by simp
qed

lemma dvd-of-dvd-mod-and-modulus:
  fixes a n d :: int
  assumes d-n: d dvd n
  assumes d-mod: d dvd (a mod n)
  shows d dvd a
proof −
  have a = n * (a div n) + a mod n
    by simp
  moreover have d dvd n * (a div n)
    using d-n by simp
  ultimately show ?thesis
    using d-mod by (metis dvd-add)
qed

lemma Gcd-one-not-all-mod-multiples-of-proper-divisor:
```

```
  fixes A :: int set
  assumes gcd-one: Gcd A = 1
  assumes d-gt-one: 1 < d
  assumes d-n: d dvd n
  assumes residues: ⋀a. a ∈ A ⟹ d dvd (a mod n)
  shows False
proof −
  have dvd-all: ⋀a. a ∈ A ⟹ d dvd a
  proof −
    fix a
    assume a-in: a ∈ A
    show d dvd a
      by (rule dvd-of-dvd-mod-and-modulus[OF d-n residues[OF a-in]])
  qed
  have d dvd (1 :: int)
    by (rule common-divisor-dvd-one-of-Gcd-one[OF gcd-one dvd-all])
  with d-gt-one show False
    by auto
qed

lemma normalized-mod-image-subset-stabilizer-contradicts-Gcd-one:
  fixes A :: int set
  assumes fin: finite A
  assumes card-ge: 3 ≤ card A
  assumes zero: 0 ∈ A
  assumes nonneg: ⋀x. x ∈ A ⟹ 0 ≤ x
  assumes gcd-one: Gcd A = 1
  assumes small-doubling: card (sumset A A) ≤ 3 * card A − 4
  assumes stable-nonempty: stable-sum-holes A ≠ {}
  defines n ≡ Max A
  defines B ≡ mod-image-int n A
  defines p ≡ nat n
  defines C ≡ Zmod.sumset p B B
  assumes B-subset-H: B ⊆ Zmod.stabilizer p C
  shows False
proof −
  have nonempty: A ≠ {}
    using zero by auto
  have top: n ∈ A
    using fin nonempty by (simp add: n-def)
  have subset: A ⊆ {0..n}
    using normalized-subset-interval[OF fin zero nonneg] by (simp add: n-def)
  have n-pos: 0 < n
  proof −
    have 1 < card A
      using card-ge by simp
    then have A ≠ {0}
      using zero fin by auto
    then obtain y where y-in: y ∈ A and y-ne: y ≠ 0
      using zero by auto
    have 0 < y
      using nonneg[OF y-in] y-ne by linarith
    also have y ≤ n
      using fin y-in by (simp add: n-def)
    finally show ?thesis .
  qed
  have p-pos: 0 < p
    using n-pos by (simp add: p-def)
  have int-p: int p = n
```

```
  using n-pos by (simp add: p-def)
have B-sub: B ⊆ {0..int p − 1}
  using mod-image-int-subset-residues[OF n-pos, of A] by (simp add: B-def int-p)
have C-eq-mod: C = mod-sumset-int n A A
proof −
  have C = mod-sumset-int (int p) B B
    unfolding C-def by (rule zmod-sumset-eq-mod-sumset-int[OF p-pos B-sub B-sub])
  also have ... = mod-sumset-int n B B
    by (simp add: int-p)
  also have ... = mod-sumset-int n A A
    unfolding B-def by (rule mod-sumset-int-mod-image-self[OF n-pos])
  finally show ?thesis .
qed
have C-sub: C ⊆ {0..int p − 1}
  using C-eq-mod mod-image-int-subset-residues[OF n-pos, of sumset A A]
  by (auto simp: mod-sumset-int-def int-p)
have zero-C: 0 ∈ C
proof −
  have witnesses: ∃ a∈A. ∃ b∈A. 0 = (a + b) mod n
  proof
    show 0 ∈ A
      by (rule zero)
    show ∃ b∈A. 0 = (0 + b) mod n
    proof
      show 0 ∈ A
        by (rule zero)
      show 0 = (0 + 0) mod n
        using n-pos by simp
    qed
  qed
  then have 0 ∈ mod-sumset-int n A A
    by (simp add: mod-sumset-int-iff)
  then show ?thesis
    using C-eq-mod by simp
qed
have H-nontrivial:
  Zmod.stabilizer p C ≠ {0}
proof −
  have Zmod.stabilizer (nat n)
      (Zmod.sumset (nat n) (mod-image-int n A) (mod-image-int n A)) ≠ {0}
    by (rule normalized-small-doubling-mod-stabilizer-nontrivial
        [OF fin card-ge n-pos subset zero top small-doubling])
  then show ?thesis
    by (simp add: B-def p-def C-def)
qed
have H-proper: Zmod.stabilizer p C ≠ {0..int p − 1}
proof
  assume full: Zmod.stabilizer p C = {0..int p − 1}
  obtain x where x-stable: x ∈ stable-sum-holes A
    using stable-nonempty by auto
  have stable-eq:
    stable-sum-holes A =
      {x. 0 < x ∧ x < n ∧ x ∉ mod-sumset-int n A A}
    by (rule stable-sum-holes-eq-mod-sumset-gaps[OF fin n-pos subset zero top])
  then have x-bounds: 0 < x x < n
    using x-stable by auto
  have x-gap: x ∉ mod-sumset-int n A A
    using x-stable stable-eq by auto
  have carrier-sub: {0..int p − 1} ⊆ C
```

```
    by (rule zmod-full-stabilizer-zero-imp-carrier-subset
        [OF p-pos C-sub zero-C full])
  have x ∈ {0..int p − 1}
    using x-bounds by (simp add: int-p)
  then have x ∈ C
    using carrier-sub by auto
  with x-gap C-eq-mod show False
    by simp
 qed
 obtain d where d-gt-one: 1 < d and d-p: d dvd int p
    and d-H: ⋀h. h ∈ Zmod.stabilizer p C ⟹ d dvd h
  using zmod-stabilizer-obtain-proper-divisor[OF p-pos H-nontrivial H-proper]
  by blast
 have d-n: d dvd n
  using d-p by (simp add: int-p)
 have residues: ⋀a. a ∈ A ⟹ d dvd (a mod n)
 proof −
  fix a
  assume a-in: a ∈ A
  have a mod n ∈ B
    using a-in by (auto simp: B-def mod-image-int-def)
  then have a mod n ∈ Zmod.stabilizer p C
    using B-subset-H by auto
  then show d dvd (a mod n)
    by (rule d-H)
 qed
 show False
  by (rule Gcd-one-not-all-mod-multiples-of-proper-divisor
      [OF gcd-one d-gt-one d-n residues])
qed

lemma Min-normalized-int-set:
 assumes fin: finite A and card-ge2: 2 ≤ card A
 shows Min (normalized-int-set A) = 0
proof −
 have nonempty: A ≠ {}
  using card-ge2 by auto
 have zero: 0 ∈ normalized-int-set A
  by (rule zero-in-normalized-int-set[OF fin nonempty])
 have nonneg: ⋀x. x ∈ normalized-int-set A ⟹ 0 ≤ x
  by (rule normalized-int-set-nonneg[OF fin card-ge2])
 show ?thesis
  by (rule Min-eq-zero-of-zero-mem-nonneg[OF finite-normalized-int-set[OF fin] zero nonneg])
qed

lemma affine-image-int-subset-ap:
 assumes A ⊆ int-ap-segment a d n
 shows affine-image-int c e A ⊆ int-ap-segment (c + e * a) (e * d) n
proof
 fix x
 assume x ∈ affine-image-int c e A
 then obtain y where y: y ∈ A and x: x = c + e * y
  by (auto simp: affine-image-int-def)
 from assms y obtain i where i-lt: i < n and y-eq: y = a + int i * d
  by (auto simp: int-ap-segment-def)
 have x = (c + e * a) + int i * (e * d)
  using x y-eq by (simp add: algebra-simps)
 with i-lt show x ∈ int-ap-segment (c + e * a) (e * d) n
  by (auto simp: int-ap-segment-def)
```

qed

4.7 Progression covers

definition *progression-cover-length-at-most* :: *int set* ⇒ *nat* ⇒ *bool* **where**
progression-cover-length-at-most A L ⟷
(∃ *a d n*. *0* < *d* ∧ *A* ⊆ *int-ap-segment a d n* ∧ *n* ≤ *L*)

lemma *progression-cover-length-at-mostI*:
assumes *0* < *d* **and** *A* ⊆ *int-ap-segment a d n* **and** *n* ≤ *L*
shows *progression-cover-length-at-most A L*
using *assms* **unfolding** *progression-cover-length-at-most-def* **by** *blast*

lemma *progression-cover-length-at-mostE*:
assumes *progression-cover-length-at-most A L*
obtains *a d n* **where** *0* < *d A* ⊆ *int-ap-segment a d n n* ≤ *L*
using *assms* **unfolding** *progression-cover-length-at-most-def* **by** *blast*

lemma *progression-cover-length-at-most-mono*:
assumes *cover*: *progression-cover-length-at-most A L*
assumes *le*: *L* ≤ *M*
shows *progression-cover-length-at-most A M*
proof −
obtain *a d n* **where** *d-pos*: *0* < *d* **and** *A-sub*: *A* ⊆ *int-ap-segment a d n* **and** *n-le*: *n* ≤ *L*
using *cover* **by** (*rule progression-cover-length-at-mostE*)
from *n-le le* **have** *n* ≤ *M*
by *simp*
with *d-pos A-sub* **show** *?thesis*
by (*rule progression-cover-length-at-mostI*)
qed

lemma *progression-cover-of-diameter-bound*:
assumes *fin*: *finite A*
assumes *nonempty*: *A* ≠ {}
assumes *len-le*: *nat* (*Max A* − *Min A* + *1*) ≤ *L*
shows *progression-cover-length-at-most A L*
proof −
have *one-pos*: *0* < (*1* :: *int*)
by *simp*
have *A-sub*: *A* ⊆ *int-ap-segment* (*Min A*) *1* (*nat* (*Max A* − *Min A* + *1*))
by (*rule finite-int-set-subset-min-max-ap*[*OF fin nonempty*])
show *?thesis*
by (*rule progression-cover-length-at-mostI*[*OF one-pos A-sub len-le*])
qed

lemma *progression-cover-affine-image-pos*:
assumes *cover*: *progression-cover-length-at-most A L*
assumes *e-pos*: *0* < *e*
shows *progression-cover-length-at-most* (*affine-image-int c e A*) *L*
proof −
obtain *a d n* **where** *d-pos*: *0* < *d* **and** *A-sub*: *A* ⊆ *int-ap-segment a d n* **and** *n-le*: *n* ≤ *L*
using *cover* **by** (*rule progression-cover-length-at-mostE*)
have *ed-pos*: *0* < *e* ∗ *d*
using *e-pos d-pos* **by** *simp*
have *image-sub*: *affine-image-int c e A* ⊆ *int-ap-segment* (*c* + *e* ∗ *a*) (*e* ∗ *d*) *n*
by (*rule affine-image-int-subset-ap*[*OF A-sub*])
show *?thesis*
by (*rule progression-cover-length-at-mostI*[*OF ed-pos image-sub n-le*])
qed

theorem *freiman-3k-minus-4-from-diameter-bound*:
assumes *fin*: *finite A*
assumes *card-ge*: *3* ≤ *card A*
assumes *diameter-le*: *nat* (*Max A* − *Min A* + *1*) ≤ *card* (*sumset A A*) − *card A* + *1*
shows *progression-cover-length-at-most A* (*card* (*sumset A A*) − *card A* + *1*)
proof −
have *A* ≠ {}
using *card-ge* **by** *auto*
then show *?thesis*
by (*rule progression-cover-of-diameter-bound*[*OF fin - diameter-le*])
qed

theorem *normalized-progression-cover-from-max-bound*:
assumes *fin*: *finite A*
assumes *card-ge*: *3* ≤ *card A*
assumes *zero*: *0* ∈ *A*
assumes *nonneg*: ⋀*x*. *x* ∈ *A* ⟹ *0* ≤ *x*
assumes *max-bound*: *nat* (*Max A* + *1*) ≤ *card* (*sumset A A*) − *card A* + *1*
shows *progression-cover-length-at-most A* (*card* (*sumset A A*) − *card A* + *1*)
proof −
have *min0*: *Min A* = *0*
by (*rule Min-eq-zero-of-zero-mem-nonneg*[*OF fin zero nonneg*])
have *nat* (*Max A* − *Min A* + *1*) ≤ *card* (*sumset A A*) − *card A* + *1*
using *max-bound* **by** (*simp add*: *min0*)
then show *?thesis*
by (*rule freiman-3k-minus-4-from-diameter-bound*[*OF fin card-ge*])
qed

theorem *normalized-progression-cover-from-hole-contribution*:
fixes *A* :: *int set*
assumes *fin*: *finite A*
assumes *card-ge*: *3* ≤ *card A*
assumes *zero*: *0* ∈ *A*
assumes *nonneg*: ⋀*x*. *x* ∈ *A* ⟹ *0* ≤ *x*
assumes *hole-cover*:
card (*interval-holes A*) ≤ *card* (*lower-sum-holes A*) + *card* (*upper-sum-holes A*)
shows *progression-cover-length-at-most A* (*card* (*sumset A A*) − *card A* + *1*)
proof −
have *max-bound*: *nat* (*Max A* + *1*) ≤ *card* (*sumset A A*) − *card A* + *1*
by (*rule normalized-max-bound-from-hole-contribution*[*OF fin zero nonneg hole-cover*])
show *?thesis*
by (*rule normalized-progression-cover-from-max-bound*[*OF fin card-ge zero nonneg max-bound*])
qed

theorem *normalized-progression-cover-from-no-stable-sum-holes*:
fixes *A* :: *int set*
assumes *fin*: *finite A*
assumes *card-ge*: *3* ≤ *card A*
assumes *zero*: *0* ∈ *A*
assumes *nonneg*: ⋀*x*. *x* ∈ *A* ⟹ *0* ≤ *x*
assumes *stable-empty*: *stable-sum-holes A* = {}
shows *progression-cover-length-at-most A* (*card* (*sumset A A*) − *card A* + *1*)
proof −
have *hole-cover*:
card (*interval-holes A*) ≤ *card* (*lower-sum-holes A*) + *card* (*upper-sum-holes A*)
by (*rule hole-cover-of-no-stable-sum-holes*[*OF stable-empty*])
show *?thesis*
by (*rule normalized-progression-cover-from-hole-contribution*

```
    [OF fin card-ge zero nonneg hole-cover])
qed

theorem normalized-progression-cover-from-short-diameter:
  fixes A :: int set
  assumes fin: finite A
  assumes card-ge: 3 ≤ card A
  assumes zero: 0 ∈ A
  assumes nonneg: ⋀x. x ∈ A ⟹ 0 ≤ x
  assumes short: nat (Max A) ≤ 2 * card A - 3
  shows progression-cover-length-at-most A (card (sumset A A) - card A + 1)
proof -
  have stable-empty: stable-sum-holes A = {}
    by (rule stable-sum-holes-empty-of-short-diameter[OF fin card-ge zero nonneg short])
  show ?thesis
    by (rule normalized-progression-cover-from-no-stable-sum-holes
        [OF fin card-ge zero nonneg stable-empty])
qed

theorem normalized-progression-cover-from-stable-hole-contribution:
  fixes A :: int set
  assumes fin: finite A
  assumes card-ge: 3 ≤ card A
  assumes zero: 0 ∈ A
  assumes nonneg: ⋀x. x ∈ A ⟹ 0 ≤ x
  assumes small-doubling: card (sumset A A) ≤ 3 * card A - 4
  assumes stable-contribution:
    stable-sum-holes A ≠ {} ⟹
      card A - 2 ≤ card (lower-sum-holes A) + card (upper-sum-holes A)
  shows progression-cover-length-at-most A (card (sumset A A) - card A + 1)
proof -
  have stable-empty: stable-sum-holes A = {}
  proof (rule ccontr)
    assume stable-sum-holes A ≠ {}
    then have extra:
      card A - 2 ≤ card (lower-sum-holes A) + card (upper-sum-holes A)
      by (rule stable-contribution)
    have lower:
      2 * card A - 1 + card (lower-sum-holes A) + card (upper-sum-holes A)
        ≤ card (sumset A A)
      by (rule normalized-sumset-lower-bound-with-holes[OF fin zero nonneg])
    have 3 * card A - 3 ≤ card (sumset A A)
      using card-ge extra lower by linarith
    with small-doubling show False
      using card-ge by linarith
  qed
  show ?thesis
    by (rule normalized-progression-cover-from-no-stable-sum-holes
        [OF fin card-ge zero nonneg stable-empty])
qed

lemma normalized-stabilizer-count:
  fixes A :: int set
  assumes fin: finite A
  assumes card-ge: 3 ≤ card A
  assumes zero: 0 ∈ A
  assumes nonneg: ⋀x. x ∈ A ⟹ 0 ≤ x
  defines n ≡ Max A
  defines B ≡ mod-image-int n A
```

```
  defines p ≡ nat n
  defines C ≡ Zmod.sumset p B B
  defines H ≡ Zmod.stabilizer p C
  defines D ≡ Zmod.sumset p B H
  assumes not-subset: ¬ B ⊆ H
  shows card H ≤
    1 + card (lower-sum-holes A ∩ upper-sum-holes A) +
      2 * (card D − card B)
proof −
  have nonempty: A ≠ {}
    using zero by auto
  have top: n ∈ A
    using fin nonempty by (simp add: n-def)
  have subset: A ⊆ {0..n}
    using normalized-subset-interval[OF fin zero nonneg] by (simp add: n-def)
  have max-eq: Max A = n
    by (simp add: n-def)
  have n-pos: 0 < n
  proof −
    have 1 < card A
      using card-ge by simp
    then have A ≠ {0}
      using zero fin by auto
    then obtain y where y-in: y ∈ A and y-ne: y ≠ 0
      using zero by auto
    have 0 < y
      using nonneg[OF y-in] y-ne by linarith
    also have y ≤ n
      using fin y-in by (simp add: n-def)
    finally show ?thesis .
  qed
  have p-pos: 0 < p
    using n-pos by (simp add: p-def)
  have int-p: int p = n
    using n-pos by (simp add: p-def)
  have B-sub: B ⊆ {0..int p − 1}
    using mod-image-int-subset-residues[OF n-pos, of A] by (simp add: B-def int-p)
  have H-sub: H ⊆ {0..int p − 1}
    unfolding H-def using Zmod.stabilizer-subset-group[of p C] p-pos by auto
  have H-sub-n: H ⊆ {0..n − 1}
    using H-sub by (simp add: int-p)
  have finB: finite B
    unfolding B-def by (rule finite-mod-image-int[OF fin])
  have zero-B: 0 ∈ B
    unfolding B-def mod-image-int-def
  proof (rule image-eqI)
    show 0 = 0 mod n
      by simp
    show 0 ∈ A
      by (rule zero)
  qed
  have B-eq: B = A − {n}
    unfolding B-def by (rule mod-image-int-normalized-interval[OF n-pos subset zero top])
  have zero-H: 0 ∈ H
    unfolding H-def by (rule Zmod.zero-mem-stabilizer)
  have add-closed-H: (x + y) mod n ∈ H if x-H: x ∈ H and y-H: y ∈ H for x y
  proof −
    have (x + y) mod int p ∈ H
      unfolding H-def by (rule zmod-stabilizer-add-closed[OF x-H[unfolded H-def] y-H[unfolded H-def]])
```

```
  then show ?thesis
    using int-p by simp
qed
have not-subset-unfolded:
  ¬ B ⊆ Zmod.stabilizer p (Zmod.sumset p B B)
  using not-subset by (simp add: H-def C-def)
obtain b c where b-B: b ∈ B and c-B: c ∈ B
    and coset-disj-unfolded:
      Zmod.sumset p {((b + c) mod int p)}
        (Zmod.stabilizer p (Zmod.sumset p B B)) ∩
        Zmod.sumset p B (Zmod.stabilizer p (Zmod.sumset p B B)) = {}
  by (rule zmod-obtain-sum-coset-disjoint-D
      [OF p-pos B-sub finB zero-B not-subset-unfolded])
have coset-disj: Zmod.sumset p {((b + c) mod int p)} H ∩ D = {}
  using coset-disj-unfolded by (simp add: C-def H-def D-def)
define R where R = mod-translate-int n b H
define S where S = mod-translate-int n c H
define K where K = mod-translate-int n ((b + c) mod n) H
have b-A: b ∈ A and b-ne-top: b ≠ n
  using b-B B-eq by auto
have c-A: c ∈ A and c-ne-top: c ≠ n
  using c-B B-eq by auto
have b-bounds: 0 ≤ b b < n
  using B-sub b-B b-ne-top by (auto simp: int-p)
have c-bounds: 0 ≤ c c < n
  using B-sub c-B c-ne-top by (auto simp: int-p)
have center-carrier: (b + c) mod int p ∈ {0..int p − 1}
  using p-pos by (auto intro!: pos-mod-bound)
have K-zmod: Zmod.sumset p {((b + c) mod int p)} H = K
  using zmod-singleton-sumset-eq-mod-translate-int[OF p-pos center-carrier H-sub]
  by (simp add: K-def int-p)
have K-disj-D: K ∩ D = {}
  using coset-disj K-zmod by simp
have K-sub-n: K ⊆ {0..n − 1}
  unfolding K-def by (rule mod-translate-int-subset-residues[OF n-pos])
have B-subset-D: B ⊆ D
proof
  fix x
  assume x-B: x ∈ B
  have x-carrier: x ∈ {0..int (p − 1)}
    using B-sub x-B p-pos by auto
  have zero-carrier: 0 ∈ {0..int (p − 1)}
    using p-pos by auto
  have (x + 0) mod int p ∈ Zmod.sumset p B H
    by (rule Zmod.sumset.sumsetI[OF x-B x-carrier zero-H zero-carrier])
  then show x ∈ D
    using x-carrier p-pos by (simp add: D-def)
qed
have K-disj-A: K ∩ A = {}
proof
  show K ∩ A ⊆ {}
  proof
    fix x
    assume x-in: x ∈ K ∩ A
    then have x-K: x ∈ K and x-A: x ∈ A
      by auto
    have x-bounds: 0 ≤ x x < n
      using K-sub-n x-K by auto
    have x-ne-top: x ≠ n
```

```
    using x-bounds by linarith
  have x-B: x ∈ B
    using x-A x-ne-top B-eq by auto
  then have x-D: x ∈ D
    using B-subset-D by auto
  with x-K K-disj-D show x ∈ {}
    by auto
 qed
qed simp
have K-disj-raw: mod-translate-int n ((b + c) mod n) H ∩ A = {}
 using K-disj-A by (simp add: K-def)
have local-count-raw:
 card H ≤
  1 + card ((lower-sum-holes A ∩ upper-sum-holes A) ∩
     mod-translate-int n ((b + c) mod n) H) +
    card (mod-translate-int n b H − A) + card (mod-translate-int n c H − A)
 by (rule sum-coset-lower-upper-inter-card
    [OF n-pos fin max-eq H-sub-n zero-H add-closed-H b-A
     b-bounds(1) b-bounds(2) c-A c-bounds(1) c-bounds(2) K-disj-raw])
have local-count:
 card H ≤
  1 + card ((lower-sum-holes A ∩ upper-sum-holes A) ∩ K) +
    card (R − A) + card (S − A)
 using local-count-raw by (simp add: R-def S-def K-def)
have D-sub: D ⊆ {0..int p − 1}
 unfolding D-def using Zmod.sumset-subset-carrier[of p B H] p-pos by auto
have finD: finite D
 using D-sub by (rule finite-subset) simp
have R-sub-D: R ⊆ D
proof
 fix x
 assume x-R: x ∈ R
 then obtain h where h-H: h ∈ H and x-eq: x = (b + h) mod n
  by (auto simp: R-def mod-translate-int-def)
 have b-carrier: b ∈ {0..int (p − 1)}
  using B-sub b-B p-pos by auto
 have h-carrier: h ∈ {0..int (p − 1)}
  using H-sub h-H p-pos by auto
 have (b + h) mod int p ∈ Zmod.sumset p B H
  by (rule Zmod.sumset.sumsetI[OF b-B b-carrier h-H h-carrier])
 then show x ∈ D
  using x-eq by (simp add: D-def int-p)
qed
have S-sub-D: S ⊆ D
proof
 fix x
 assume x-S: x ∈ S
 then obtain h where h-H: h ∈ H and x-eq: x = (c + h) mod n
  by (auto simp: S-def mod-translate-int-def)
 have c-carrier: c ∈ {0..int (p − 1)}
  using B-sub c-B p-pos by auto
 have h-carrier: h ∈ {0..int (p − 1)}
  using H-sub h-H p-pos by auto
 have (c + h) mod int p ∈ Zmod.sumset p B H
  by (rule Zmod.sumset.sumsetI[OF c-B c-carrier h-H h-carrier])
 then show x ∈ D
  using x-eq by (simp add: D-def int-p)
qed
have R-minus-sub: R − A ⊆ D − B
```

proof
fix x
assume *x-in*: $x \in R - A$
then have *x-D*: $x \in D$
using *R-sub-D* **by** *auto*
have $x \notin B$
using *x-in B-eq* **by** *auto*
then show $x \in D - B$
using *x-D* **by** *simp*
qed
have *S-minus-sub*: $S - A \subseteq D - B$
proof
fix x
assume *x-in*: $x \in S - A$
then have *x-D*: $x \in D$
using *S-sub-D* **by** *auto*
have $x \notin B$
using *x-in B-eq* **by** *auto*
then show $x \in D - B$
using *x-D* **by** *simp*
qed
have *finDminus*: *finite* $(D - B)$
using *finD* **by** *simp*
have *card-R-minus*: *card* $(R - A) \leq$ *card* $(D - B)$
by (*rule card-mono*[*OF finDminus R-minus-sub*])
have *card-S-minus*: *card* $(S - A) \leq$ *card* $(D - B)$
by (*rule card-mono*[*OF finDminus S-minus-sub*])
have *card-inter*:
card $((lower\text{-}sum\text{-}holes\ A \cap upper\text{-}sum\text{-}holes\ A) \cap K) \leq$
card $(lower\text{-}sum\text{-}holes\ A \cap upper\text{-}sum\text{-}holes\ A)$
proof (*rule card-mono*)
show *finite* $(lower\text{-}sum\text{-}holes\ A \cap upper\text{-}sum\text{-}holes\ A)$
by *simp*
show $lower\text{-}sum\text{-}holes\ A \cap upper\text{-}sum\text{-}holes\ A \cap K \subseteq$
$lower\text{-}sum\text{-}holes\ A \cap upper\text{-}sum\text{-}holes\ A$
by *auto*
qed
have *diff-card*: *card* $(D - B) =$ *card* $D -$ *card* B
by (*rule card-Diff-subset*[*OF finB B-subset-D*])
have *card* $H \leq$
$1 +$ *card* $(lower\text{-}sum\text{-}holes\ A \cap upper\text{-}sum\text{-}holes\ A) +$
$2 *$ *card* $(D - B)$
using *local-count card-R-minus card-S-minus card-inter* **by** *linarith*
then show *?thesis*
using *diff-card* **by** *simp*
qed

theorem *normalized-progression-cover-from-stabilizer-count*:
fixes A :: *int set*
assumes *fin*: *finite* A
assumes *card-ge*: $3 \leq$ *card* A
assumes *zero*: $0 \in A$
assumes *nonneg*: $\bigwedge x.\ x \in A \Longrightarrow 0 \leq x$
assumes *gcd-one*: *Gcd* $A = 1$
assumes *small-doubling*: *card* $(sumset\ A\ A) \leq 3 *$ *card* $A - 4$
defines $n \equiv Max\ A$
defines $B \equiv mod\text{-}image\text{-}int\ n\ A$
defines $p \equiv nat\ n$
defines $C \equiv Zmod.sumset\ p\ B\ B$

```
  defines H ≡ Zmod.stabilizer p C
  defines D ≡ Zmod.sumset p B H
  assumes stabilizer-count:
    stable-sum-holes A ≠ {} ⟹
      ¬ B ⊆ H ⟹
      card H ≤
        1 + card (lower-sum-holes A ∩ upper-sum-holes A) +
        2 * (card D − card B)
  shows progression-cover-length-at-most A (card (sumset A A) − card A + 1)
proof −
  have stable-empty: stable-sum-holes A = {}
  proof (rule ccontr)
    assume stable-nonempty: stable-sum-holes A ≠ {}
    have nonempty: A ≠ {}
      using zero by auto
    have top: n ∈ A
      using fin nonempty by (simp add: n-def)
    have subset: A ⊆ {0..n}
      using normalized-subset-interval[OF fin zero nonneg] by (simp add: n-def)
    have n-pos: 0 < n
    proof −
      have 1 < card A
        using card-ge by simp
      then have A ≠ {0}
        using zero fin by auto
      then obtain y where y-in: y ∈ A and y-ne: y ≠ 0
        using zero by auto
      have 0 < y
        using nonneg[OF y-in] y-ne by linarith
      also have y ≤ n
        using fin y-in by (simp add: n-def)
      finally show ?thesis .
    qed
    have p-pos: 0 < p
      using n-pos by (simp add: p-def)
    have int-p: int p = n
      using n-pos by (simp add: p-def)
    have B-sub: B ⊆ {0..int p − 1}
      using mod-image-int-subset-residues[OF n-pos, of A] by (simp add: B-def int-p)
    have finB: finite B
      unfolding B-def by (rule finite-mod-image-int[OF fin])
    have zero-B: 0 ∈ B
      unfolding B-def mod-image-int-def
    proof (rule image-eqI)
      show 0 = 0 mod n
        by simp
      show 0 ∈ A
        by (rule zero)
    qed
    have nonemptyB: B ≠ {}
      using zero-B by auto
    have cardB: card B = card A − 1
      unfolding B-def
      by (rule card-mod-image-int-normalized-interval[OF fin n-pos subset zero top])
    have C-eq-mod: C = mod-sumset-int n A A
    proof −
      have C = mod-sumset-int (int p) B B
        unfolding C-def by (rule zmod-sumset-eq-mod-sumset-int[OF p-pos B-sub B-sub])
      also have ... = mod-sumset-int n B B
```

```
    by (simp add: int-p)
  also have ... = mod-sumset-int n A A
    unfolding B-def by (rule mod-sumset-int-mod-image-self[OF n-pos])
  finally show ?thesis .
qed
have sum-card:
  card (sumset A A) =
    card C + card A + card (lower-sum-holes A ∩ upper-sum-holes A)
  using normalized-sumset-card-eq-mod-sumset-plus-card-inter-holes
    [OF fin n-pos subset zero top - nonneg]
  by (simp add: n-def C-eq-mod)
show False
proof (cases B ⊆ H)
  case True
  have B-subset-exact:
    mod-image-int (Max A) A ⊆
      Zmod.stabilizer (nat (Max A))
        (Zmod.sumset (nat (Max A))
          (mod-image-int (Max A) A) (mod-image-int (Max A) A))
    using True by (simp add: n-def B-def p-def C-def H-def)
  show False
    by (rule normalized-mod-image-subset-stabilizer-contradicts-Gcd-one
        [OF fin card-ge zero nonneg gcd-one small-doubling stable-nonempty
         B-subset-exact])
next
  case False
  have count:
    card H ≤
      1 + card (lower-sum-holes A ∩ upper-sum-holes A) +
      2 * (card D − card B)
    by (rule stabilizer-count[OF stable-nonempty False])
  have H-def': H = Zmod.stabilizer p C
    by (simp add: H-def)
  have D-def': D = Zmod.sumset p B H
    by (simp add: D-def)
  have kneser-raw:
    2 * card (Zmod.sumset p B
        (Zmod.stabilizer p (Zmod.sumset p B B))) −
      card (Zmod.stabilizer p (Zmod.sumset p B B))
      ≤ card (Zmod.sumset p B B)
    by (rule zmod-kneser-self[OF p-pos B-sub finB nonemptyB])
  have kneser:
    2 * card D − card H ≤ card C
    using kneser-raw by (simp add: C-def H-def D-def)
  have D-ge-B: card B ≤ card D
  proof −
    have B-subset-D: B ⊆ D
    proof
      fix b
      assume b-in: b ∈ B
      have b-carrier: b ∈ {0..int (p − 1)}
        using B-sub b-in p-pos by auto
      have 0 ∈ H
        unfolding H-def by (rule Zmod.zero-mem-stabilizer)
      moreover have 0 ∈ {0..int (p − 1)}
        using p-pos by auto
      ultimately have (b + 0) mod int p ∈ Zmod.sumset p B H
        by (rule Zmod.sumset.sumsetI[OF b-in b-carrier])
      then show b ∈ D
```

```
        using b-carrier p-pos by (simp add: D-def H-def)
      qed
      have finD: finite D
      proof −
        have D ⊆ {0..int (p − 1)}
          unfolding D-def by (rule Zmod.sumset-subset-carrier)
        then have D ⊆ {0..int p − 1}
          using p-pos by auto
        then show ?thesis
          by (rule finite-subset) simp
      qed
      show ?thesis
        by (rule card-mono[OF finD B-subset-D])
    qed
    have C-lower:
      2 ∗ card B − 1 − card (lower-sum-holes A ∩ upper-sum-holes A)
        ≤ card C
    proof −
      have card H ≤
          1 + card (lower-sum-holes A ∩ upper-sum-holes A) +
          2 ∗ card D − 2 ∗ card B
        using count D-ge-B by linarith
      with kneser D-ge-B show ?thesis
        by linarith
    qed
    have 3 ∗ card A − 3 ≤ card (sumset A A)
      using sum-card C-lower cardB card-ge by linarith
    with small-doubling card-ge show False
      by linarith
  qed
 qed
 show ?thesis
  by (rule normalized-progression-cover-from-no-stable-sum-holes
      [OF fin card-ge zero nonneg stable-empty])
qed

theorem normalized-progression-cover-from-gcd-one:
 fixes A :: int set
 assumes fin: finite A
 assumes card-ge: 3 ≤ card A
 assumes zero: 0 ∈ A
 assumes nonneg: ⋀x. x ∈ A ⟹ 0 ≤ x
 assumes gcd-one: Gcd A = 1
 assumes small-doubling: card (sumset A A) ≤ 3 ∗ card A − 4
 shows progression-cover-length-at-most A (card (sumset A A) − card A + 1)
proof −
 define n where n = Max A
 define B where B = mod-image-int n A
 define p where p = nat n
 define C where C = Zmod.sumset p B B
 define H where H = Zmod.stabilizer p C
 define D where D = Zmod.sumset p B H
 have count:
   card H ≤
     1 + card (lower-sum-holes A ∩ upper-sum-holes A) +
       2 ∗ (card D − card B)
   if stable-nonempty: stable-sum-holes A ≠ {} and not-subset: ¬ B ⊆ H
 proof −
   have not-subset-raw:
```

```
    ¬ mod-image-int (Max A) A ⊆
      Zmod.stabilizer (nat (Max A))
        (Zmod.sumset (nat (Max A))
          (mod-image-int (Max A) A) (mod-image-int (Max A) A))
    using not-subset by (simp add: n-def B-def p-def C-def H-def)
  show ?thesis
    unfolding n-def B-def p-def C-def H-def D-def
    by (rule normalized-stabilizer-count[OF fin card-ge zero nonneg not-subset-raw])
 qed
 show ?thesis
 proof (rule normalized-progression-cover-from-stabilizer-count
    [OF fin card-ge zero nonneg gcd-one small-doubling])
  assume stable-nonempty: stable-sum-holes A ≠ {}
  assume not-subset-raw:
    ¬ mod-image-int (Max A) A ⊆
      Zmod.stabilizer (nat (Max A))
        (Zmod.sumset (nat (Max A))
          (mod-image-int (Max A) A) (mod-image-int (Max A) A))
  show card
      (Zmod.stabilizer (nat (Max A))
        (Zmod.sumset (nat (Max A))
          (mod-image-int (Max A) A) (mod-image-int (Max A) A)))
    ≤ 1 + card (lower-sum-holes A ∩ upper-sum-holes A) +
      2 * (card
        (Zmod.sumset (nat (Max A)) (mod-image-int (Max A) A)
          (Zmod.stabilizer (nat (Max A))
            (Zmod.sumset (nat (Max A))
              (mod-image-int (Max A) A) (mod-image-int (Max A) A)))) −
        card (mod-image-int (Max A) A))
    using count[OF stable-nonempty] not-subset-raw
    by (simp add: n-def B-def p-def C-def H-def D-def)
 qed
qed

theorem freiman-3k-minus-4-from-normalized-core:
 assumes core:
  ⋀B. finite B ⟹
    3 ≤ card B ⟹
    0 ∈ B ⟹
    (⋀x. x ∈ B ⟹ 0 ≤ x) ⟹
    Gcd B = 1 ⟹
    card (sumset B B) ≤ 3 * card B − 4 ⟹
    progression-cover-length-at-most B (card (sumset B B) − card B + 1)
 assumes fin: finite A
 assumes card-ge: 3 ≤ card A
 assumes small-doubling: card (sumset A A) ≤ 3 * card A − 4
 shows progression-cover-length-at-most A (card (sumset A A) − card A + 1)
proof −
 let ?N = normalized-int-set A
 let ?g = int-set-content A
 have card-ge2: 2 ≤ card A
   using card-ge by simp
 have nonempty: A ≠ {}
   using card-ge by auto
 have finN: finite ?N
   by (rule finite-normalized-int-set[OF fin])
 have cardN: card ?N = card A
   by (rule card-normalized-int-set[OF fin card-ge2])
 have sumN: card (sumset ?N ?N) = card (sumset A A)
```

```
    by (rule card-sumset-normalized-int-set[OF fin card-ge2])
  have cardN-ge: 3 ≤ card ?N
    using card-ge cardN by simp
  have zeroN: 0 ∈ ?N
    by (rule zero-in-normalized-int-set[OF fin nonempty])
  have nonnegN: ⋀x. x ∈ ?N ⟹ 0 ≤ x
    by (rule normalized-int-set-nonneg[OF fin card-ge2])
  have gcdN: Gcd ?N = 1
    by (rule Gcd-normalized-int-set[OF fin card-ge2])
  have smallN: card (sumset ?N ?N) ≤ 3 * card ?N - 4
    using small-doubling cardN sumN by simp
  have coverN:
      progression-cover-length-at-most ?N (card (sumset ?N ?N) - card ?N + 1)
    by (rule core[OF finN cardN-ge zeroN nonnegN gcdN smallN])
  have g-pos: 0 < ?g
    by (rule int-set-content-pos[OF fin card-ge2])
  have cover-image:
      progression-cover-length-at-most
        (affine-image-int (Min A) ?g ?N)
        (card (sumset ?N ?N) - card ?N + 1)
    by (rule progression-cover-affine-image-pos[OF coverN g-pos])
  have A-eq: A = affine-image-int (Min A) ?g ?N
    using normalized-int-set-reconstruction[OF fin card-ge2] by simp
  have length-eq:
      card (sumset ?N ?N) - card ?N + 1 =
       card (sumset A A) - card A + 1
    using cardN sumN by simp
  from cover-image A-eq have
      progression-cover-length-at-most A (card (sumset ?N ?N) - card ?N + 1)
    by metis
  with length-eq show ?thesis
    by metis
qed
```

4.8 The target statement

The AFP-facing statement combines the normalization lemmas above with the additive-combinatorial core proof: a finite integer set A with $card\ (sumset\ A\ A) \leq 3 * card\ A - 4$ and $3 \leq card\ A$ is contained in an arithmetic progression of length at most $card\ (sumset\ A\ A) - card\ A + 1$.

```
theorem freiman-3k-minus-4:
  fixes A :: int set
  assumes fin: finite A
  assumes card-ge: 3 ≤ card A
  assumes small-doubling: card (sumset A A) ≤ 3 * card A - 4
  shows progression-cover-length-at-most A (card (sumset A A) - card A + 1)
proof (rule freiman-3k-minus-4-from-normalized-core[OF - fin card-ge small-doubling])
  fix B :: int set
  assume finB: finite B
  assume cardB-ge: 3 ≤ card B
  assume zeroB: 0 ∈ B
  assume nonnegB: ⋀x. x ∈ B ⟹ 0 ≤ x
  assume gcdB: Gcd B = 1
  assume smallB: card (sumset B B) ≤ 3 * card B - 4
  show progression-cover-length-at-most B (card (sumset B B) - card B + 1)
    by (rule normalized-progression-cover-from-gcd-one
        [OF finB cardB-ge zeroB nonnegB gcdB smallB])
qed
```

end

References

[1] M. B. Nathanson. *Additive Number Theory: Inverse Problems and the Geometry of Sumsets*, volume 165 of *Graduate Texts in Mathematics*. Springer-Verlag, 1996.

[2] T. Tao and V. H. Vu. *Additive Combinatorics*, volume 105 of *Cambridge Studies in Advanced Mathematics*. Cambridge University Press, 2006. DOI: https://doi.org/10.1017/CBO9780511755149.